C言語
実用マスター
シリーズ 3

「改訂 新C言語入門」

スーパービギナー編

著
林晴比古

SOFT BANK
Publishing

林 晴比古の
言語入門ラインアップ

改訂 新C言語入門 ビギナー編〈好評既刊〉

改訂 新C言語入門 シニア編〈好評既刊〉

改訂 新C言語入門 スーパービギナー編〈本書〉

改訂 新C言語入門 応用編〈好評既刊〉

●

新C++入門〈好評既刊〉

●

新Visual C++6.0入門 ビギナー編〈好評既刊〉

新Visual C++6.0入門シニア編〈好評既刊〉

新Visual C++6.0入門スーパービギナー編〈好評既刊〉

新Visual C++5.0入門 ビギナー編〈好評既刊〉

新Visual C++5.0入門 シニア編〈好評既刊〉

●

新Visual Basic入門 ビギナー編 Ver.6.0対応〈好評既刊〉

新Visual Basic入門 シニア編 Ver.6.0対応〈好評既刊〉

新Visual Basic入門 ビギナー編 Ver.5.0対応〈好評既刊〉

新Visual Basic入門 シニア編 Ver.5.0対応〈好評既刊〉

Visual Basic応用プログラミング Ver.5.0/6.0対応〈好評既刊〉

●

C/C++によるプログラミングスタイルブック〈好評既刊〉

カバーデザイン/火取ユーゴ

C言語
実用マスター
シリーズ
3

「改訂 新 C言語入門」

スーパービギナー編

著
林晴比古

●お知らせ

本書掲載の主なプログラム（ソースコード）は、インターネットの次のURLアドレスに登録される予定になっていますので、ご利用ください。

SOFTBANK ONLINE BOOKLIST（ソフトバンク書籍のページ）

http://books.softbank.co.jp/

まえがき

C言語はコンピュータプログラミングの世界でもっとも重要なものです。通産省の情報処理技術者試験においても、C言語が試験対象に採用されています。同じ試験採用言語にCOBOLやFORTRANもありますが、現在このCOBOLとFORTRANが主力言語であると考える人はいないでしょう。今やコンピュータプログラミングの世界では、C言語を知らないとどうにもならないという時代になっているのです。

Windowsの世界ではグラフィック操作が多用されますが、その開発言語としてC++系統のものが用いられます。C++はCの機能を強化したもので、このC++を学習したい場合にも、「Cの知識」が必須となります。このように、Cはさまざまなプログラム開発シーンにおける基幹言語としての地位をもっています。

本書はそのC言語について、まったくの初心者に基礎的な知識を伝えるために書かれたものです。

本来C言語は多少なりとも、プログラミングに関する知識をもっている人が「次に学ぶ」ものでした。初心者はもっと構造の簡単な言語（その代表例はBASIC）で、プログラミングに関する素養を身に付け、その知識を土台にしてC言語を学習するというのが効率的な方法です。ですから、現実にC言語の本の中で、

——変数とは何か

について解説することはないのが普通でした。まず簡単な言語に取り組み、それをクリアし、「次はCに挑戦」となるのが無理のない道なのです。

しかし、以前は高価であったパソコンが身近なものになり、同じように、昔は高価であったC言語処理系が安価に手に入るようになると、

——Cのことを手短に知りたい

という要求が出されるようになってきました。その目的は、

——Cとはどういうものか一般知識程度は知っておきたい

とか、

——Cを本格的に学ぶべきかどうか検討したい

とか、いろいろあります。また指導的立場の人で、

——自分でCを使うことはないが、他人にCを語るための知識を得たい

という場合もあるでしょう。つまり形態は違っていても「いきなりCに挑戦」という人たちが登場してきているのです。

そのような人たちに、Cを理解してもらうにはどうすればいいのだろうか、というのは結構重いテーマです。しかし説明次第で、かなりのところまで理解することは可能な気がします。少なくともCの骨組みになる部分は理解可能です。そして実用プログラムを書くための知識も修得できます。

この目的を達成するためには、「とりあえず必要でないことは覚えない」という見切りが

必要です。たとえば英会話を例にとると、相手の話を聞くために、原則的には、

——英語の全文法、全単語のマスター

が必要です。なぜなら、相手はどんな文法で、どんな単語で話しかけてくるかわからないからです。しかし一方、相手に何かの意思を伝えることは比較的容易です。それは俗にいうカタコト英会話でいいのです。なぜなら、自分の知っている文法と単語だけを使って話せばいいからです。

Cを学ぶときにも、まずはこの「カタコトC言語のススメ」を試みてみるのは無駄ではありません。他人の書いたCプログラムリストを読もうとすると、C言語の総合力を必要とします。しかし他人のプログラムにかまうことなく、ただ自分で使うプログラムを自分で作るだけなら、少ない知識（すなわちカタコトC言語）で十分です。ポインタなどまったく使わなくても、実用的なプログラムを作るのは可能です。そしてカタコトC言語に慣れたら、次のステップとして少しずつボキャブラリを増やしていけばいいのです。

本書のシリーズである「新C言語入門 ビギナー編」「新C言語入門 シニア編」は、本格的にC言語をマスターするためのステップ1、ステップ2になっています。そこではC言語の機能が順序正しく、もれることなく説明されています。それに対して本「スーパービギナー編」は、この2冊とは視点を変えて、もっとリラックスした雰囲気でC言語を体験する内容になっています。いってみれば、「C言語セミナー紙上1日コース」という感じでしょうか。そもそも、「ビギナー編でもまだ難しい」という読者の声に触発されて書かれたのが本書なのです。

このように、C言語修得の第1段階をクリアすることを目的としてこの本は書かれました。重要な機能を厳選し、サンプルプログラムを基に説明を展開し、前提知識をさほど必要とせず、難しい話をなくし、図解を多くして、やわらかな雰囲気の中で、狭い範囲ながらも体系的に知識が修得できるように配慮してあります。本書は「新C言語入門 ビギナー編」の前段階の本、あるいは副読本としても有益ですが、C言語の手短な理解のための独立した1冊として、本書だけを楽しむこともできます。C言語とはどんなものだろうと興味をもった方や、C言語を少しやってみようかと思った方にとって、本書が一助となることを願っています。

●改訂新版に際して

本書は1992年に初版を発行いたしました。幸い非常に好評で、1998年まで順調に刷りを重ねてきました。今回、「新C言語入門」シリーズ（スーパービギナー編、ビギナー編、シニア編）を一括改訂する目的は、時代にそぐわなくなってきた事実（たとえばTurbo C、Quick Cなどの処理系の変遷）、コンピュータ環境の進化（UNIXは現在でも健在だが、MS-DOSがWindowsに置換された）、そして言語環境の発展（32ビットint型が主となった）などとの整合をとることが主です。併せて、内容を見直し、理解を深めるための補足記述を入れ、図版を多くし、入門書としてより充実させることを目指しました。

1998年8月8日　林 晴比古

CONTENTS

CONTENTS

CONTENTS

Chapter 1

C とはどのような言語か

What's "C"?

1.1 C言語の登場

Cは1972年ごろ、アメリカのベル研究所で設計されたプログラミング言語です。はじめは主にUNIXというオペレーティングシステム（OS）を開発するために使用されました。

そのCには特徴がたくさんありますが、重要なポイントの1つとして言語仕様のコンパクトさがあります。同じプログラミング言語である、COBOL、FORTRAN、PL/Iなどに比べ、C処理系はとても小さいのです。そのため資源のとぼしいコンピュータ環境でも走らせることができ、また移植がしやすいために、広く普及していきました。現在ではパーソナルコンピュータやワークステーション上の主力言語となり、また大型コンピュータにまでその利用範囲を広げています。

もう1つCが普及した理由として、アセンブラに近いきめ細かい記述ができ、実行速度の速いプログラムを開発できることがあります。従来の高級言語はアセンブラに比べて標準パターンの記述は容易になるものの、マシン環境に介入するような記述（すなわちアセンブラ的な仕事）は苦手で、できたプログラムの実行速度も劣るものでした。つまり、

　　書きやすさか、スピードか

という二者択一をともなうものだったのです。昔のコンピュータはプログラム開発環境が不十分だったため、

──スピードは多少落ちてもいいから、プログラムを書きやすいようにしてほしい

というのがユーザーの一番の要求でした。そしてスピードが落ちたとしても、それでも手作業よりは格段に速いので、それほど不満はなかったのです。また、「高級言語とはそんなもの」という認識も定着していました。

ところがコンピュータ利用が活発になるにつれて、

──プログラムが書きやすいことはもちろんだが、スピードももっと上げてほしい

という要求がでるようになりました。

Cはこの問題を改善しました。Pascalに似た雰囲気の記述容易性（これが高級言語本来の目的です）をもちながら、比較的細かいことまで記述でき、比較的実行速度の速いプログラムを開発することができます。

最近のCコンパイラはさらに洗練されてきて、従来アセンブラでしか記述できなかった割込処理なども記述でき、そして最適化技術の向上により、アセンブラに比べて遜色ないスピードを実現しています。

1.2 Cで何ができるか

Cはそもそもシステムプログラム開発用の言語です。つまりUNIXやMS-DOSなどのオペレーティングシステムや、コンパイラ、リンカ、エディタ、そして各種のツールプログラムの開発に便利なように設計されています。たとえばCでは1文字単位で何かの処理をするための機能が強力です。これはシステムプログラム開発のためには重要なことです。一方、COBOLやBASICは、1文字単位の処理は苦手です。というより重要視していません。それはこれらの言語の目的が異なるからです。

このようにCは特別の目的のために設計されました。しかし最近では便利な標準関数がたくさん用意され、BASICやCOBOLやFORTRANでできることは皆できるようになってきています。そこで現在では、

——Cは汎用のプログラミング言語である

といわれるようになっています。

Cを使えば日本語ワープロソフトや表計算ソフトなどのアプリケーションプログラムから、CADプログラム、科学技術計算プログラム、そしてゲームプログラムなど、さまざまなプログラムを開発することができます。

1.3 Cプログラミングのためには何が必要か

手元にパソコンのある読者が、C言語を使ってみたいと思いたったとき、何が必要になるでしょうか。基本的な構成としては次のようになります。

(1) OS（MS-DOS、Windows、UNIXなど）
(2) エディタ
(3) Cコンパイラ

パソコン上で走るC言語は、それぞれのオペレーティングシステムの上で走るようになっています。パソコンを使う場合、そのOSはMacintoshなど特別のものを除けば、Windows（またはMS-DOS）が一般的です。ただしWindowsを使う場合も、C言語そのものはMS-DOS上で利用するのが普通です。そのためWindowsの場合も、

Windows上の［MS-DOSプロンプト］

を起動して、「Windows上に作られたMS-DOS環境」でC言語を使うことができます。今後これを「MS-DOSウィンドウ」と呼ぶことにします。

次にプログラムを作成するためには「エディタ」と呼ばれるプログラムが必要です。これにはMS-DOSの時代から使われてきたMIFESやVZエディタ（いずれも商品名です）などがあります。またMS-DOSには標準でEDITというエディタが用意されていますか

ら、機能は多少限定されますがこれを使用してもかまいません。ソースプログラムを入力できればいいわけですから、MS-DOSウィンドウを使う場合には、何か他のエディタ系のプログラムを使ってもかまいません。たとえば多少不便ではありますが、Windows上のアクセサリである「メモ帳」を使うことも可能です。

また、かつてよく使われたTurbo CやQuick Cなどのように、コンパイラとエディタが一体になったC処理系もあります。Windows上でよく使われるVisual C++という処理系も、エディタ一体型になっています。このような処理系を使用するときは、わざわざエディタを別途購入する必要はありません。

1.4 C処理系の２つの環境

現在市販されているC処理系は2つの環境をもっています。それを仮に「個別開発環境」および「統合開発環境」と呼ぶことにします。

個別開発環境はその名の通り、エディタ、コンパイラ、そしてデバッガなどを別々に操作するものです。たとえばMS-DOS環境の上で、MIFESというエディタでソースプログラムを作り、それをMicrosoft Cでコンパイルして実行プログラムを得るためには次の操作をします。

```
C>mifes smp1.c        ……smp1.cプログラムを記述

C>cl smp1.c           ……smp1.cプログラムをコンパイル

C>smp1                ……作成されたsmp1.exeプログラムを実行する
```

一方、統合開発環境の場合は、まずその処理系を起動します。するとエディタを備えた画面構成になります。その一体化された画面の中で、プログラムを入力し、コンパイルし、実行します。

次に示すのはWindows用の言語であるMicrosoft Visual C++という処理系を使ってsmp1.cを作成し、実行している画面です。エディタも一体になった統合開発環境といえます。

Microsoft Visual C++の統合開発環境の画面

これをクリックするとコンパイルする

ここをクリックするとプログラムを実行する

ソースプログラムを入力する

コンパイル結果

実行すると自動的にMS-DOSウィンドウが開かれ、smp1.exeが起動される

またこのMicrosoft Visual C++は、Microsoft CというC言語処理系も内蔵していて、それをMS-DOSウィンドウで使用することができます。次の例はMS-DOSウィンドウで作業中の画面です。エディタ起動や、コンパイル指示を個別に指定する個別開発環境になります。

Visual C++を使ってコンパイルを行なうまでの操作は「Appendix　Visual C++を利用してCコンパイラを使う方法」(183ページ)で説明しています。

MS-DOSウィンドウで**smp1.c**プログラムを作成する

「mifes smp1.c」を入力してエディタを使用中の画面

```
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ - MIFES
C:¥CWORK¥SMP1.C          [   5:  20]      <挿入>        前面+F1=ヘルプ
#include <stdio.h>
main()
{
    printf("Hello C¥n");
}
[EOF]
1ﾒﾆｭｰ1 2ﾒﾆｭｰ2 3ﾒﾆｭｰ3 4画面割 5検索↓6行選択7カット8コピー 9ﾍﾟｰｽﾄ 0タグJP
```

実行プログラム**smp1.exe**を実行する

作成したsmp1.cをコンパイルしてsmp1.exeを得る

```
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ
C:¥CWORK>cl smp1.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright (C) Microsoft Corp 1984-1997. All rights reserved.

smp1.c
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.

/out:smp1.exe
smp1.obj

C:¥CWORK>smp1
Hello C

C:¥CWORK>_
```

できたsmp1.exeを起動する　　正しく表示された

1.5 ソフトウェアを準備する

パソコンを所有していて、これからC言語の学習をしたいと思ったら、必要なソフトウェアを用意しなければなりません。それにはまずCコンパイラを購入し、それをパソコンに組み込むインストール作業が必要です。

Windowsが登場してから、市販されるソフトウェアはCD-ROMで供給されることが普通になりました。この場合、そのCD-ROMを、通常の操作でCD-ROMドライブに挿入すると、自動的にインストール開始画面になります（「オートプレイ機能」といいます）。後は画面に表示される指示に従うだけです。

純粋なMS-DOSマシンの場合は、オートプレイ機能がありませんから、CD-ROMまたはFDを挿入したら、キーボードからインストール用のコマンドを入力することになります。そのコマンドはマニュアルに指定してありますが、たいていは、

```
install
```

または、

```
setup
```

と入力します。この後画面上に表示される質問に答えることにより、インストール作業を進めることができます。それぞれの質問に際しては、適切なアドバイスや説明が付きます。たいていのインストールプログラムでは質問に対する標準的な答えが示されていますので、よくわからなければ「標準」を選択すれば、無難な設定でインストール作業を成功させることができます。インストールが終わったら、購入したフロッピーディスクやCD-ROMは大切に保管しておきます。不手際や思わぬアクシデントで、再度インストールすることもあるからです。

いずれにせよ、インストールの方法は購入した製品のマニュアルなどに書かれていますから、それに従うことになります。

1.6 ハードウェアとソフトウェアを理解する

コンピュータはハードウェアとソフトウェアの協力により仕事をしています。両者が緊密な関係を維持することにより、コンピュータは整然と動作をしてくれるのです。ですから両者の関係を多少は知っておくと、Cコンパイラの理解の助けになります。

ソフトウェアの指示

ハードウェアとは、コンピュータを構成する機器類のことで、鉄や、プラスチックや、ICチップやプリント基板などで構成されています。コンピュータは、本体以外にもディスプレイや外部ディスクやプリンタなどが必要です。これらも皆ハードウェアと呼ばれています。ハードウェアはとても重要ですが、それだけでは何の仕事もしません。コンピュータに仕事をさせるためにはソフトウェア、すなわちプログラムが必要です。

ハードウェアはあらかじめ決められた何種類もの仕事を、高速で行なう優れた能力をもっています。しかしその能力をどう組み合わせて使えば、意味のある仕事ができるのかについては知りません。その指示を行なうのがソフトウェアです。ソフトウェアは、

——これこれの順番でこれこれの動作をしろ

という指示をだします。ハードウェアはそれらの与えられた指示を忠実に実行しますが、1つ1つの動作がどのような役目をもっているのかは知りません。それでもいわれた通りに動作を行なうと、結果として意味のある仕事をすることになるのです。

機械語

ハードウェアは、この「これこれの動作」というもっとも基本的なレベルの仕事を行なうようにできています。その「これこれの動作」は機械語と呼ばれるもので指示します。機械語だけがハードウェアに直接働きかけられるわけです。機械語で指示されたことだけをハードウェアは理解します。ここでいう機械語とは仮に「語」と呼んでいるだけで、実体は信号線に流れる信号の組み合わせを意味します。その組み合わせに従って何かの動作をするわけです。

たとえばコンピュータ本体の中に、命令を判断するための信号線が8本あるとします。この8本のどこに信号が流れるかによって、基本的な動作が行なわれます。8086系MPU(Micro Processing Unit：パソコンの頭脳にあたる部分で各種の演算を行なうもの)で例をあげてみましょう。次の図の1は「信号あり」、0は「信号なし」を意味します。

信号の組み合わせ

信号線7　信号線6　信号線5　信号線4　信号線3　信号線2　信号線1　信号線0

信号　0　1　0　0　1　0　0　0

このような組み合わせをもつ信号が流れたら、8086系MPUでは、「AXレジスタの内容を−1する」という仕事をします。レジスタとは「数値を記憶する部品」と思ってください。これがコンピュータ本体が行なうことのできる働きの最小単位になります。

このような信号の組み合わせはたくさんあります。理屈の上では8本の信号線で256通り（2の8乗）の異なった指示をだすことができます。それ以上の命令をだすためには、この信号線グループを2組、3組と組み合わせることにより実現します。

ビットとバイト

この0/1を表現できる1つの信号（信号線1本に相当します）を「ビット」という単位で呼んでいます。ここでは信号が8つありますから、8ビットの情報になります。そして8ビットの情報をまとめて1バイトと呼んでいます。

1ビット＝1つの0/1情報
1バイト＝8ビット

ビットとバイト

1ビットで0または1を表現

0　1　0　0　1　0　0　0

1ビット

全部で8ビット

1バイトと呼ぶ

16進数で表現

これが機械語と呼ばれるものです。この機械語の内容を表現するときは、どういえばいいのでしょうか。そのまま、

ゼロイチゼロゼロ イチゼロゼロゼロ

と呼ぶのは、あまりにも無策というものです。そこで信号が1になっている場所を表現するために4つの信号ずつに区切って、「0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F」と数える方法が工夫されました。この場合「0100 1000」は「48」と表現します。この数え方を「16進数」といいます。

信号線を16進数で数える

16進数の数え方を次に示します。

16進数の数え方

呼び方	信号の組み合わせ
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

※数え方の例

01001000 → 4 8

10110011 → B 3

アセンブラ命令

16進数表現はビット単位で「ゼロイチゼロゼロ、イチゼロゼロゼロ」と呼ぶよりは便利ですが、それでも単なる数字表現であり、表意的ではありません。すなわち「48」という16進数の数字から、

1だけ減じる

という意味を読み取ることはできません。そこで機械語と1対1に対応しながら、表意的な呼称を与える方法が考えだされました。それがアセンブラと呼ばれるプログラミング方法です。アセンブラもプログラミング言語の1つに数えられています。

この場合は「48」という数字に対して、

DEC

というアセンブラ命令が付けられています。つまり「48」に対して「DEC」という愛称を与えたわけです。このDECはdecrement（減少）から付けられた名前です。こうすることに

より「減じる」という意味がわかりやすくなります。アセンブラではその他、

```
MOV   AX,DATA1        ;AXにDATA1を入れる (move:動かす)
ADD   AX,5            ;AXを +5 する       (add:加える)
MOV   DATA2,AX        ;DATA2にAXを入れる
```

といった記述をします。アセンブラが開発されたことにより、機械語よりはわかりやすいプログラミングが可能になりました。

高級言語

しかしそれでも約束ごとの細かすぎる記述であることには変わりありません。そこでもっと自然な表記が可能になるプログラミング言語が開発されました。これが高級言語と呼ばれるものです。COBOL、FORTRAN、BASIC、Pascal、Cなどは皆、高級言語です。高級言語(たとえばC)では上のアセンブラプログラムを、

```
data2=data1+5;
```

と書きます。これは我々の数学表記に近いので、容易に処理内容を理解できます。

Cコンパイラはこのように書かれたもの(ソースプログラムといいます)を機械語に変換します。これを「コンパイルする」といいます。最終的に機械語に変換されたものをコンピュータ本体は理解して仕事をしているわけです。

コンパイラによる変換

この関係は日本人とアメリカ人が意思疎通することを考えるとよく理解できます。次の図で理解してください。

コンピュータとの意思疎通

●アメリカ人との意思疎通

日本語の文書ができる
翻訳者が日本語→英語の翻訳をする
英語の文書ができる
アメリカ人がそれを理解する

日本人：日本語しかわからない
アメリカ人：英語しかわからない

●コンピュータとの意思疎通

C言語の文書を作成
コンパイラがC言語→機械語の翻訳をする
機械語の文書ができる
コンピュータがそれを理解する

```
main()
{
  int a;
  a=5;
  .......
```

```
01000010
0011....
```

OK!

ユーザー：C言語を勉強した
コンピュータ：機械語しかわからない

1.7 プログラミング言語の選択

このようにコンピュータに何か仕事を行なわせるためにはソフトウェアの助けが必要になります。すなわちプログラムを作ってハードウェアに指示を与えなければなりません。そのプログラムはプログラミング言語を用いて作ります。そのプログラミング言語にはいくつもの種類があります。なぜいくつもあるかというとまず、

――プログラミングの目的別

というものがあります。たとえば事務処理を行ないたい場合、事務処理特有の手順というものがあります。したがってその特有の処理形態をサポートした言語を事務処理用プログラムの記述に用いると便利です。具体的にはCOBOLがそれに該当します。また科学技術計算のためにはさまざまな数学関数を必要とします。したがってその種の関数の完備しているFORTRANを使ったほうが、技術計算は楽になります。次に、

――流儀の違い

というものがあります。同じ目的の言語であっても、「どのような仕様になっていれば使いやすいか」ということに関しては、設計者の主張があります。それによってまたいろいろな言語が登場します。LISPやPrologという言語は特殊な用途に用いられます。Cはその中の1つの言語です。Cは汎用のプログラミング言語として使うことができ、その主張がマーケットで受け入れられたため広く普及しています。

1.8 必要なことだけ覚える

そのプログラミング言語を覚えるときに、ぜひアドバイスしたいことがあります。それはとりあえず不要なものに興味をもつなということです。たとえばC言語では、

```
printf("abcde");
```

とすると、画面に"abcde"と表示してくれます。このとき、

――なぜこうすると画面に表示されるのだろうか

と考える必要はありません。それを追究するとコンパイラの仕組みを理解し、機械語を理解し、MPUの仕組みを理解し、ディスプレイの仕組みを理解しなければならなくなります。そんなことをしていたら、いくら時間があっても足りません。これからプログラミング言語を学ぼうとする初心者にそんな余裕はないのです。新米プログラマにハードウェアのメカニズムを理解することを強制することには問題があります。

――ハードに首を突っ込むな

というアドバイスは重要です。たとえば、

――AをしたらBになる

というソフトウェア上の事実があった場合、

――なぜAをしたらBになるのだろう

ということを考え込んでいたら大変です。仕事が進みません。プログラマとしては、

――なぜかは知らないけれどAをしたらBになるらしい。ではこれを利用して何かおもしろいことをしよう

と考えるのが正しい姿勢なのです。

よくいわれる説で、

――Cのポインタを理解するためにはハードウェアのことを知らなければならない

というのがあります。これはハードウェアというものを総括的にとらえるなら間違いではありません。しかしこれも、必要なのは「アドレス指定のメカニズム」を知ることであり、それは実は機械語というソフトウェアの分野における重要項目である「アドレッシングの方法」を知ればいいことなのです。

厳密にいえば、もちろんハードウェアのことを理解する必要のあるケースもあるでしょう。しかし一般傾向としてプログラマは、ソフトのことだけを考えていないと、時間不足になります。

ハードに首を突っ込むな

printf("abcde");

abcde

なぜこうなるのだろうか？

・コンパイラの仕組みの勉強
・MPUの勉強
・ディスプレイの勉強
・その他の勉強…

迷宮入り…

1.9 プログラムを構成するもの

プログラムはいろいろな要素で構成されます。これらを理解しないとプログラムを作ることはできません。Cプログラムを構成する主要なものをあげると次のようになります。

Cプログラムの構成要素

変数
定数
コメント
キーワード
識別子
構造体
ポインタ
配列
式
プリプロセッサ
演算子
一般文
制御文
関数

　これらの要素を組み合わせることによってプログラムが作成されます。このときこれらの要素を覚えようとして暗記する必要はありません。これらはすべて、
　――必要だから用意された
　ものです。ですから逆にその必要性を実感できれば、自然に理解できるのです。たとえばプログラミング「言語」というからには、あらかじめ決められた単語があるはずです。それが、

```
if   for   while   int   float
```

といったキーワードになります。またプログラムの中に注釈を入れることができなければ不便でたまりません。そのため、

```
/*        */
```

というコメント機能があります。また一時的に値を保存できないと不便です。そこで、

　　変数

という機能があります。さらに変数に対して、

```
a=a+10;
```

といった計算ができなければ仕事になりません。そこで、

```
+ - * /
```

といった演算子（この場合は算術演算子）が用意されています。そして、
——ある数字が10以下だったら○○をやる。そうでなかったら××をやる
という切り分け仕事もできないと微妙な処理ができません。そのために、

```
if else
```

といった制御文があります。さらに、
——○○の処理を繰り返せ
といった記述も必要で、そのために、

```
for while
```

といった制御文があります。これらはすべて必要だから用意されているわけで、暗記して覚えようというものではありません。だからプログラマも、必要性を感じたらその用法を理解すればいいわけです。たとえばポインタを使う必要性というものを、何も感じないプログラマは、ポインタを無理に覚える必要はありません。ポインタを使わなくても十分にプログラムを作ることはできます。そしてもっと上達したときに、

——こういうことができればいいのに

と強く考えることがあるかもしれません。それがポインタの利用で一挙に解決するのであれば、そこでポインタについて勉強すればいいのです。

業務でプログラミングするのでなければ、C言語を学ぶのに、早すぎることも遅すぎることもありません。マイペースでかまわないのです（もし業務でC言語を必要とする人は、基本的な機能を1か月で使えるようにならないと、有能とは見なされないでしょう）。

とにかくプログラムの構成要素は、それぞれ合理的な理由があって用意されていることを理解しましょう。

Chapter 2

どうやってプログラムを作るか

2.1 ソースプログラムの作成

ではプログラムはどうやって作ればいいのでしょうか。それにはまず、
——どのような流派(言語)を使うか
を決めなければいけません。それがC言語と決定したら次に、
——Cという言語での話のしかた
を覚えなければなりません。プログラミング言語といっても、相手に意思を伝えるという意味では日本語や英語や中国語と同じものです。それぞれに単語をもっていて文法をもっているだけです。ただしプログラミング言語は「～しろ」という指示語(すなわち命令)だけからなる少々いばった言語です。
それでも言語であることには変わりありませんから、相手が理解できる言葉で話しかけてやれば意思は通じます。たとえば、

「**abcde**」と画面に表示しろ

と日本語でいってもコンピュータは理解できません。コンピュータには日本語という言語はわからないからです。しかしコンピュータはC言語は理解できます(実際にはC言語から変換された機械語を理解できるということですが)。ですからC言語の文法を使って、

```
printf("abcde");
```

としてやれば、コンピュータは画面に「abcde」と表示してくれるのです。このように「Cという言語での話のしかた」を使って、プログラムを書いてやります。これがC言語プログラミングです。
たとえていうと、町には英語を学ぶために「英会話学校」があります。これと同じように、C言語を学ぶために「C会話学校」があっても不思議ではない、そのようなものです。

どちらも言語

●日本語

"abcde"と表示しろ

理解できない…

●C言語

printf("abcde");

理解できる

abcde

プログラムはワープロで手紙を書くのと同じように、キーボードから文字を入力して、プログラムとしてのファイルを作ります。ただしプログラムは半角入力を多用しますから、日本語ワープロでは効率がよくありません。そこでエディタと呼ばれるプログラムを使います。

統合開発環境をもっているC処理系の場合は、その標準エディタを使います。個別開発環境のC処理系の場合は、MIFESやEDITのような独立したエディタを使います。

このようにエディタで作成されたソースプログラム（ソースファイルともいいます）は、

・アルファベットを主体に書かれている
・C言語のルールで書かれている
・基本的に文字だけで構成されていて、罫線などは含まれない

といったこと以外は、通常の日本語ワープロで作成された文書ファイルと同じものです。

このCソースプログラムは、まだCコンパイラとは無関係です。単にCコンパイラに対応した記述をした文字の集まりにすぎません。Cコンパイラをもっていない人でも、Cソースプログラムを書くことはできます。Cコンパイラをもっているということと、Cソースプログラムを書くということは別の話です。

約束によりCソースプログラムをファイルに保存するときは.cというタイプ名を付けます。たとえば、

```
tstprg.c
```

という名前を付けます。

2.2 ソースプログラムをコンパイルする

ソースプログラムを作成しても、それだけでコンピュータが仕事をしてくれるわけではありません。何度も述べたように、コンピュータが理解できる唯一の言語である機械語に変換してやらなければならないのです。その仕事をしてくれるのがCコンパイラです。CコンパイラはCソースプログラムを読み込み、ある処理単位ごとにそれを機械語に変換していきます。コンパイル処理の実行は統合開発環境の場合、プルダウンメニューの中の「コンパイル」に相当する項目を選択することにより行なわれます。また個別開発環境の場合は、MS-DOSやUNIXのコマンドラインからコンパイルを指示します。

例として`tstprg.c`をコマンドラインからコンパイル指示する方法は次のようになります。Turbo CやQuick Cは現在ではあまり用いられていませんが、歴史的に重要なので示しておきます。

コンパイル指示の方法

処理系	コンパイル指示	コメント
Turbo C	C>tc tstprg.c	
Quick C	C>qc tstprg.c	
Microsoft C	C>cl tstprg.c	
UNIX 例1	% cc tstprg.c -o tstprg.exe	標準のCコンパイラ
UNIX 例2	% gcc tstprg.c -o tstprg.exe	最近のCコンパイラ（gcc）の例

Cソースプログラムが無事に機械語に変換されると、MS-DOS上のCコンパイラの場合、タイプ名が`.exe`となるファイルが新たに作成されます。UNIXの場合は指定したタイプ名になります。この例の場合は"`tstprg.exe`"というファイルが作成されます。ただしUNIXでは拡張子を"`.exe`"にする必要はありませんので、通常は、

```
% cc tstprg.c -o tstprg
```

と指定します。この場合は"`tstprg`"という名前の実行ファイルが作成されます。

できあがった実行ファイルは機械語に変換されていますので、MS-DOSなどのコマンドラインからそのまま実行することができます。または統合開発環境の中で実行させることができます。

例としてMIFESというエディタを使ってソースプログラム`tstprg.c`を作成し、それをMicrosoft Cでコンパイルして実行するという開発の流れを次に示します。

Cプログラム作成の手順

C>mifes tstprg.c	C>cl tstprg.c	C>tstprg
MIFES で ソースプログラムtstprg.cを作る	tstprg.c をコンパイルし tstprg.exeを作る	できた tstprg.exeを実行する

tstprg.c → コンパイル → tstprg.exe → 実行

なおプログラムをコンパイルしたら、次にリンクという作業が必要です。リンクでは標準ライブラリ（標準関数が入っている）を結合するなどの仕事をします（標準関数については後述）。しかし、たいていのC処理系ではコンパイル指示コマンドを用いることにより、

コンパイル　＋　リンク

という仕事を自動的に行なってくれます。したがって初心者の場合には、特別な事情がない限り、リンクを個別に実行することはありません。

2.3　エラーがでたらどうするのか

ソースプログラムをコンパイルしたとき、C言語のルールに合わない記述があるとエラーが表示されます。エラーは、

――○○行で××というエラーが発生した

という形式で報告されます。次のプログラムは、

```
printf("abcde");
```
……正しい

と書くべきところを、

```
printf("abcde";
```
……')'がない

と書いてしまったプログラムをコンパイルした場合のエラーメッセージの例です。

リスト エラーのあるプログラム：smp2.c

```
#include <stdio.h>
main()
{
    printf("abcde";
}
```

エラーメッセージ例1

```
smp2.c(4) : error C2143: syntax error : missing ')' before ';'
```

エラーメッセージ例2

```
smp2.c(4) : error C2143: 構文ｴﾗｰ : ')' が ';' の前に必要です。
```

エラーメッセージ例3

```
Error smp2.c 4: Function call missing ) in function main
```

エラーメッセージの文章はCコンパイラによってまちまちです。このようなエラーがでた場合は、その行数とエラーメッセージを読みます。そしてエラーの原因を理解し、ソースプログラムの該当行をエディタで修正します。正しく修正したらもう一度コンパイルします。これを繰り返し、エラーなしでコンパイルが終了したら、そのプログラムを実行することができます。

Microsoft C/C++によるエラー発生例

ファイル内容を見る
')'が不足している

```
C:¥CWORK>type smp2.c
#include <stdio.h>
main()
{
    printf("abcde";
}

C:¥CWORK>cl smp2.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright (C) Microsoft Corp 1984-1997. All rights reserved.

smp2.c
smp2.c(4) : error C2143: 構文ｴﾗｰ : ')' が ';' の前に必要です。

C:¥CWORK>_
```

コンパイル実行
エラー発生

2.4 警告メッセージもある

コンパイルすると、エラーとは別に「警告(Warning)」というものも表示されることがあります。たとえば、

```
#include <stdio.h>
main()
{
    int a;                  ……変数aを宣言したが
    printf("abcde");        ……それは使われていない
}
```

というプログラムがあります。これをあるC処理系でコンパイルすると、

```
Warning tst.c 6: 'a' declared but never used in function main
```

という警告がでます。これは、
　――変数aが宣言されているが一度も使われていない
　ということです。警告メッセージは、
　――エラーではないけれど、よくない書き方なので一応報告しておきます
　という機能をもっています。

NOTE

警告メッセージは、たいていの場合、そのまま放置してもプログラムは正しく動作する。そのためCコンパイラ側では、むやみに警告メッセージをださないようにしている。それでも上級プログラマは、市販アプリケーションのような厳格な品質のプログラムを開発するとき、詳細な警告メッセージを得たい場合がある。そこでたいていのCコンパイラでは、警告のレベルを設定できるようになっている。例としてMicrosoft C/C++では次の設定ができる。

警告のレベル設定

警告レベル	内容
/W0	すべての警告メッセージを無効にする
/W1	重大な警告メッセージを表示する（デフォルトの設定）
/W2	中レベルの警告メッセージも表示する
/W3	軽レベルの警告メッセージも表示する
/W4	微細な警告メッセージも情報として表示する

指定例：　**cl /W3 smp.c**

2.5 デバッグする

　コンパイルが正しく行なわれても、それでプログラム作成が終了したわけではありません。コンパイルされたプログラムは、とりあえず実行できますが、

　　プログラマが意図した通りに動作するか

という確認を必要とします。この作業を、

　　デバッグ

といいます。デバッグ作業で問題があったら、エディタでその部分を修正し、再度コンパイルし、そして動作を確認します。この作業を繰り返します。意図した通りに動くこ

とが確認されたとき、プログラムは完成したことになります。
　これら一連の開発作業の流れを図示すると次のようになります。

プログラム開発の流れ

START
エディタでソースプログラムを作成（修正）する
コンパイルする
エラー発生したか？
YES
NO
動作に問題あるか？
YES
NO
実行ファイル完成

2.6 完成したプログラムの実行

　無事に完成して実行ファイルができると、そのプログラムを実行させることができます。たとえばMS-DOS上で`tstprg.exe`という実行ファイルができているとします。現在の画面が、

```
C>
```

というプロンプト表示になっているとき、

```
C>tstprg
```

と入力すると、`tstprg.exe`が実行されます。また統合開発環境形式のC処理系の場合は、適切なメニューを指定することで、実行させることができます。

Chapter 3
Cプログラミングの8つのステップ

Cでプログラミングを行なうときに、まず学んでほしい項目は次の8つです。

8つの基本ステップ

変数宣言
入力
演算
条件判断
繰り返し
関数
出力
ファイル処理

この中でも重要なものは、変数宣言、入力、演算、条件判断、繰り返し、出力、の6項目です。これだけでとりあえずプログラムを組めるようになります。その後関数とファイル処理を覚えると、実用的なプログラムが書けるようになります。

1. 変数宣言

プログラムの中ではさまざまなデータ（情報）を使います。データを加えたり引いたり比較したりすることにより、何か有益な処理を行なわせるわけです。そのデータは、処理中にどこかに一時的に置いておかなくてはなりません。その置く場所を確保するのが「変数宣言」です。

2. 入力

変数宣言を行なって、データの置き場所を用意したら、そこによそからデータをもってくることができます。もってくる場所は通常はキーボードです。このようにデータをプログラムの外からもってきて、所定の場所に入れることを「入力」といいます。具体的にはキーボードから数値や文字列を入力することになります。もちろんキーボード以外（たとえばファイル）から入力することもあります。

3. 演算

入力されたデータは、演算することにより目的を果たします。演算のために演算子と呼ばれるものが用意されています。たとえば、データaとデータbの値を加えたものをデータcとして得たいときは、

```
c=a+b;
```

という記述を行ないます。ここで'+'は加算を行なう演算子です。

4. 条件判断

演算は、

――データをすべて加算する

といった単純なものとは限りません。たとえば、

――データをすべて加算する。ただし1000未満のものについては加算しない

という処理をしたいときには、何らかの条件判断が必要になります。この場合その条件判断は、

――もしデータの値が1000以上なら加算する

ということになります。このような条件付きの処理を行なわせるために条件文が用意されています。

5. 繰り返し

演算においてもう1つ特徴的なことに、同じ処理を何回も繰り返すことがあります。たとえば生徒が500人いて、その平均点を計算する場合、まず500人の点数の合計をとらなければなりません。そのとき単純に加算処理を500回分（500行分）記述するというのは手間のかかる話です。ここは何らかの方法で、

――500回だけ加算処理を繰り返せ

という指示をしたほうが簡単です。それを実現してくれるのが繰り返し文です。

6. 関数

大きなプログラムを書いていると、同じ処理があちこちに何回もでてくることがあります。連続してでてくるときは、繰り返し処理で記述できますが、そうではなく散発的に何度もでてくるケースです。この場合、その「よく使われる処理」を一度だけ記述し、それに名前を付けておき、別の場所において、

――あのいつもの処理をここで行なえ

と指示することが有効です。名前を付けておけば、プログラム中のどこからでもその名前を指定するだけで、その名前で表現された一連の処理を実行することができます。このような機能をサブルーチンと呼ぶこともありますが、Cでは「関数」と呼んでいます。

7. 出力

データを使って演算を行なったら、その結果を確認しなければなりません。そのためたとえばディスプレイに値を表示します。これを「出力」といいます。出力処理を行なうことによって、コンピュータ本体の内部で行なわれた目に見えない仕事の結果を、目に見えるものとして得ることができるようになります。また出力はプリンタに対して行なうこともあります。

8. ファイル処理

　キーボードからデータを入力すると、もしデータ入力ミスがあった場合、はじめからデータを入力し直すことになります。またデータをディスプレイに出力すると、画面表示が変わったり、あるいはパソコンの電源を落としたりしたとき、もうその結果を見ることができなくなります。Aというプログラムの出力した結果を、Bというプログラムの入力データとして使いたいときにも、なんらかの仕掛けがないと不都合が生じます。

　これを解決するにはプログラムの処理結果を、ファイルとして保存してやることが有効です。プログラムのデータ入力をファイルから行なうようにすれば、何度も同じデータを使って処理を行なうことができます。

　このような仕事をファイル処理といいます。ファイル処理ではファイルへのデータ書き込みと、ファイルからのデータ読み込みを行ないます。これによりデータの再利用が可能になります。

Chapter 4

まず何か出力させてみよう（出力の方法）

4.1 文字列を出力する（その1）

新しいプログラミング言語の勉強をはじめるとき、最初に覚えることは「どうやってメッセージを出力するか」ということです。出力の方法がわからなければ、そのプログラムが、正しく動作しているかどうかも確認できません。ここでは最初のプログラムとして画面に、

How are you?

という文字列を表示させてみましょう。プログラムは次のようになります。

リスト　画面に"How are you?"と表示させる

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("How are you?¥n");
6: }
```

注：行番号は説明のために付けたものなので実際に入力する必要はない。

実行結果

```
How are you?
```

Cは最小のプログラムを書く場合も、決められた枠組みを必要とするので、やや段取り仕事のイメージがあります。なお、このCプログラムリストの中で使われている行番号は、

説明のために付加した行番号

ですから注意してください。実際に皆さんがこのプログラムを入力する場合、行番号を打ち込む必要はありません。本書では以降も「説明のための行番号」を付けることがありますので了解してください。

mainは関数の名前

mainは関数の名前です。Cは「関数」というプログラム単位をユーザーが作っていく

ことで構成されます。Cの関数は他のプログラミング言語で使われる「サブルーチン」や、値を返す「関数」と同じものです。Cでは両者をまとめて、ただ「関数」と呼んでいます。関数の中にはこれから説明する変数宣言や実行文が記述されます。何かを画面に出力したり、何かをキーボードから入力したり、四則演算をしたりするための記述はすべて、どれかの関数の中に含まれなければなりません。この例は単純なプログラムなので、関数が1つ、つまりmainしかありません（厳密にはprintfも標準関数と呼ばれる関数ですが、これはユーザーが作る関数ではありません）。

関数には好きな名前を付けることができますが、1つだけ決まっているものがあります。それが「main」です。この「main」だけは特別な名前です。プログラムが複数の関数で構成されていると、どの関数から起動したらいいのかわかりません。そのため最初に起動してほしい関数にはmainという名前を付ける約束になっているのです。このようにmainは、

――Cプログラムはまずmain関数から実行される

という特別の意味をもっています。Cプログラムの中には1個のmain関数が必要です。このプログラムは小さいのでmain関数だけで構成されています。

関数の有効範囲

このプログラムの中でmain関数の有効範囲は、

4行目の'{'から
6行目の'}'まで

です。関数は名前の次に書かれた'{'から実行開始し、最後の'}'に出会うと仕事を終了します。Cでは{}はブロック文と呼ばれて盛んに使用されます。たとえばforやwhileなどの制御文の有効範囲を示すときにも{}が使われます。

空行は無視される

2行目は空行になっています。Cコンパイラは空行を単に無視します。プログラマは読みやすさのために空行を自由に入れることができます。

大文字と小文字は区別される

Cプログラムは通常、小文字で書きます。小文字で書いたほうがプログラムは見やすくなります。大文字は特別の目的のあるときに使われます。Cでは小文字と大文字は区別されます。たとえば、

```
data1
DATA1
```

はまったく異なる2つの名前になります。

printfで出力する

printfはメッセージを出力するための標準関数です。標準関数はC処理系がはじめから準備してくれている関数です。printfはいろいろ複雑な機能をもっていますが、もっとも基本的な機能としては、" "で囲まれた文字列の出力があります。

```
printf("abcdef¥n");
```

(abcdef¥n ── これを画面出力する)

¥nは復改を意味する

printf関数の出力文字列の中に¥nがあると、それは"¥n"という文字列ではなく、復改(復帰改行)という働きをします。¥nを書くことにより"How are you?"と出力した後に復改を行なってくれます。Cのprintfではプログラマが明確に指示する場合にだけ復改が行なわれます。

復改は、入れることも入れないことも、複数入れることも自由です。いろいろな復改の使用例をあげてみます。

リスト　例1

```
printf("How ");           ……¥nがない
printf("are ");           ……¥nがない
printf("you?¥n");
```

実行結果

```
How are you?
```

リスト　例2

```
printf("ABC¥nDEF¥nGHI¥n");        ……いくつも¥nがある
```

実行結果

```
ABC
DEF
GHI
```

リスト 例3

```
printf("ABC¥n¥nDEF¥nGHI¥n");          ……¥n¥nがある
```

実行結果

```
ABC
              ……2つ目の¥nによる空行
DEF
GHI
```

NOTE

復帰と改行と復改

復改と改行は同じ意味で用いられることが少なくないが、厳密には復帰、改行、復改(=復帰改行)という言葉はそれぞれに別の機能をもっている。たとえば文字列"**ABCDEFG**"を出力し、そのときのカーソルが①の位置にあったとする(①②③は位置を示すために付した)。

```
ABCDEFG①
②      ③
```

この時点でカーソルは文字'**G**'の次の位置でチカチカしている。ここで「復帰」を行なうと、カーソルは同じ行の先頭である文字'**A**'のところにくる。また「改行」すると、カーソルは単に行を進めて③の位置にくる。そして「復改」をするとカーソルは次の行の先頭、つまり②の位置にくる。

#includeで読み込む

プログラムの最初の行にある、

```
#include <stdio.h>
```

は、stdio.hという名前のファイルをこの位置に読み込む(include:含む)ことを指示しています。これを、

——stdio.hをインクルードする

と表現します。この結果stdio.hというファイルの中に書かれていることが、このサンプルプログラムの中でも有効になります。

これはstdio.hというファイルの内容と同じことを、"#include <stdio.h>"と書いている位置に、そっくり記述することと同じ効果になります。

stdio.hはCコンパイラを購入すると、必ず用意されているファイルです。入出力のための有効な記述がたくさん書かれています。stdio.hをインクルードしないと正しく動作しない関数もあります。したがって初心者はその内容を理解するしないにかかわら

ず当面は、

——プログラムの先頭には「#include <stdio.h>」を必ず書く

と覚えてください。なお、このような".h"というタイプ名の付いているファイルのことを「ヘッダファイル」と呼びます。

文にはセミコロン';'が必要

Cプログラムは関数で構成されますが、その関数はいくつもの「文」で構成されます。文は終端に';'をともなっています（ただしforやwhileといった制御文は';'を付けません）。このセミコロンが、1つの文の終わりを示します。たとえばBASICでは、

```
A=100
B=C+300
```

のように書きますが、Cでは';'を付けて、

```
a=100;
b=c+300;
```

と書きます。

文にはセミコロンを付ける

BASIC: A = 100 これでよい

C言語: a = 100 ; これも必要

字下げをすると読みやすい

5行目のprintf（34ページ）は先頭に4文字分空白をもっています。これは読みやすくするために行なわれています。これを字下げといいます。たとえば繰り返し文は、

```
for(i=1; i<=100; i++){
sum=sum+i;
```

```
mul=mul*i;
}
```

とするより、

```
for(i=1; i<=100; i++){
     sum=sum+i;
     mul=mul*i;
}
```

としたほうがfor文の制御範囲がよくわかります。字下げの数は4文字や8文字がよく用いられます。Cコンパイラはこのような空白がいくつ連続していても、これを1個の空白と同じと見なします。プログラマは読みやすさのために、自由に空白文字を入れることができます。

空白を取る

逆にCプログラムでは、余計な空白や改行を取ってもかまいません。このプログラムを最小構成にすると次のようになります。これでも正しく動作しますが、読みやすさの点ではよいプログラムとはいえません。

```
#include <stdio.h>
main(){printf("How are you?¥n");}
```

注:#ではじまる記述は単独行に書く約束になっている。

4.2 文字列を出力する(その2)

文字列を出力する方法は1つではありません。ルールに従うなら次のような記述も可能です。これはいずれも画面に、

```
今日はよい天気です.
```

と表示させるものです。

リスト　画面に「今日はよい天気です.」と表示する

```
 1: /* sample program */
 2: #include <stdio.h>
 3:
 4: main()
 5: {
 6:     printf("今日はよい天気です.¥n");
 7:
 8:     printf("今日は");
 9:     printf("よい天気");
10:     printf("です.¥n");
11:
12:     printf("今日は"); printf("よい天気"); printf("です.¥n");
13:
14: }
```

実行結果

```
今日はよい天気です.
今日はよい天気です.
今日はよい天気です.
```

漢字の出力

Cでは二重引用符の中に書かれた漢字はそのまま出力されます。それ以外の場所では漢字（全角文字）は使用できません。

1行に複数の文を書く

12行目の記述のように、Cでは1行に複数の文を書くことができます。Cの文は終端に';'をもっているため、この終端記号をチェックすることにより文の切れ目を識別できるのです。ただしCでは通常は1行1文でプログラムを作っていきます。そのほうが読みやすいからです。

1行に複数の文を書くことができる

C言語: a=10 ; b=20 ; c=30 ;

これがあるから1行にいくつ文を書いても混乱しない

コメント

1行目はコメントです。コメントは /* ではじまり */ で終わる範囲指定型です。コメントは、

```
printf("今日はよい天気です.¥n"); /* メッセージ出力例 */
```

のように文の後によく入れられますが、

```
/*
   コメント
   コメント
   コメント
*/
```

のように、複数行にまたがってまとめて入れることもできます。

コメントの開始と終了

コメント開始 → コメント終了

/*　　*/

私たちのことは無視してください

範囲指定型なので次の記述も有効です。この場合、出力は"ABCGHI"になります。

```
printf("ABC"); /*printf("DEF");*/ printf("GHI");
```

4.3 その他の出力

ここではまず文字列の出力方法を説明しましたが、この他にもいろいろな出力があります。出力させるデータ区分としては、

- 文字列の出力
- 1文字の出力
- 特殊文字の出力（¥nなど）
- 数字の出力

などがあります。ここで説明した以外の出力方法については「Chapter7　データをキーボードから入力させよう（入力）」（→67ページ）で説明します。

4.4 例題プログラム

例題1

画面に、次のように表示するプログラムを作る

```
a
ab
abc
abcd
```

リスト 解答

```
#include <stdio.h>

main()
{
    printf("a¥n");
    printf("ab¥n");
    printf("abc¥n");
    printf("abcd¥n");
}
```

例題2

同じ出力を1つのprintfで作る。コメントとしてプログラム名「printtst.c」を書く

リスト 解答

```
/* printtst.c */
#include <stdio.h>

main()
{
    printf("a¥nab¥nabc¥nabcd¥n");
}
```

この章のまとめ

- Cプログラムは関数で構成される
- 関数には好きな名前を付けることができる
- Cプログラムにはmain関数が必要である
- 関数の有効範囲は'{'から'}'までである
- 空行は無視される
- 大文字と小文字は区別される
- Cプログラムは基本的に小文字で書く
- printfはメッセージを出力するための標準関数である
- ¥nは復改を意味する
- プログラムの先頭には「#include <stdio.h>」を必ず書く
- 文にはセミコロン';'が必要である
- 読みやすくするために字下げをするとよい
- 二重引用符の中では漢字を使うことができる
- Cでは１行に複数の文を書くことができる
- コメントは /* ではじまり */ で終わる

Chapter 5

データを置いておく場所を確保しよう（変数）

5.1 変数とメモ用紙

プログラムはデータを処理することにより仕事を進めます。データとは基本的には数値と文字列です。数値と文字列をいろいろ加工することにより、何か有効な仕事をしてくれます。

データ
数　値
文字列

通常、プログラムはたった1つのデータを使用しているわけではありません。たくさんのデータをもっています。しかし一度に処理できるデータは1個だけです。

たとえば10個のデータを使用するプログラムがあります。このプログラムはある瞬間にはたった1つのデータを処理しているだけです。するとその間、残りの9個のデータはどこかにしまっておかなくてはなりません。そのしまっておく場所が「変数」といわれるものです。10個のデータを使用するときは10個の変数（格納場所）を用意し、それぞれについて、

　　変数のデータを取りだし、加工し、元に戻す

という処理をしています。変数はしばしば、

　　変数　＝　データを入れる箱

と説明されます。10個のデータを所有するためには10個の箱が必要になります。

この変数という概念は、実は日常生活でも知らないうちに使用しています。たとえば、

　　23 × 45 ＋ 67 × 89 はいくらか

という問題があります。これを筆算で解くには次のようにするはずです。

①「23 × 45」を計算する
② その答え1035をどこかにメモする
③「67 × 89」を計算する
④ その答え5963をどこかにメモする
⑤ 上の②でメモしていた数値と④でメモしていた数値を加える
⑥ 答え6998を得る

この手順のうち②と④に相当するものが変数処理です。最初の23 × 45という計算結果を、どこかにメモして保存しておかないと、次の計算に移ることができません。そのデータの保存をするために「メモ用紙」というメモリを使用しているのです。

5.2 変数と電卓

もう1つの変数のよい例は電卓にあります。一般に電卓には数字キーや加減乗除キーなどのほかに、次のようなキーが付いています。

MC　memory clear ▶ メモリに記憶されている数値をクリアする
M+　memory plus ▶ 画面に表示されている数値をメモリに記憶されている数値に加える
M-　memory minus ▶ 画面に表示されている数値をメモリに記憶されている数値から減じる
MR　memory read ▶ メモリに記憶されている数値を呼びだす

この電卓で先ほどの計算を行なうには次のキー操作をします。

MC ▶ あらかじめメモリ内容を0にしておく

2 3 × 4 5 M+ ▶ 最初の乗算をして結果をメモリに加える

6 7 × 8 9 M+ ▶ 次の乗算をして結果をメモリに加える

MR ▶ メモリ内容（答え）を読む

この中の「メモリ」と呼んでいる機能が、実は変数機能に相当します。この電卓は1個の変数をもっていると考えることができます。技術計算用の高級電卓には、このメモリが2つ付いているものもあります。これは変数が2つあることになり、それだけ込み入った計算が可能になります。たとえば変数が2つあれば、

12 × 34 ＋（55 × 66 ＋ 77 × 88）÷ 99 はいくらか

という計算もメモ用紙を使わないでできるのです。

メモリ付き電卓

5.3 変わる数だから変数

変数には好きな名前を付けることができます。例として3つの変数(箱)を用意し、それぞれ、a、b、cという名前を付けたとしましょう。変数操作の例とそのときどきの変数の値を示すと次のようになります。

変数の操作例

操作	記述例	操作後の箱の内容
変数a、b、cを用意する	`int a,b,c;`	a: (空)　b: (空)　c: (空)　最初の値は不定
a、b、cの内容を0にする	`a=0;b=0;c=0;`	a: 0　b: 0　c: 0
aに10を入れる	`a=10;`	a: 10　b: 0　c: 0
bに40を入れる	`b=40;`	a: 10　b: 40　c: 0
aに20を加える	`a=a+20;`	a: 30　b: 40　c: 0
aとbを加えたものをcに入れる	`c=a+b;`	a: 30　b: 40　c: 70
cの内容を表示する	`printf("%d¥n",c);`	a: 30　b: 40　c: 70　→ 70　画面に70を表示

この例でわかるように箱の内容(数)は一定しないで、どんどん変化します。そのため「変数」と呼ばれています。この他に、値が一定していて変化しない数というのもあり、それは「定数」と呼ばれています。

5.4 変数の値を出力する

次のプログラムは変数を確保し、それにデータを入れ、そのデータを加算処理します。そして加算結果をprintfで画面に表示します。

リスト 「10+20」の答えを計算して表示する

```
 1: #include <stdio.h>
 2:
 3: main()
 4: {
 5:      int a,b,c;
 6:
 7:      a=10;
 8:      b=20;
 9:      c=a+b;
10:      printf("%d¥n",c);
11: }
```

実行結果

```
30
```

整数型変数の宣言

5行目は変数宣言です。

```
int a,b,c;
```

は「int型の変数、a、b、cをこれから使用する」ということを宣言するものです。int型(integer：整数)の変数とは「整数を入れる箱」のことです。また「変数を宣言する」とは、

――これこれの変数を使うので準備をしてくれ

とコンパイラに知らせることです。これによりコンパイラは3つの整数の値を格納する場所を設定し、それぞれが、a、b、cという名前で使えるように準備をします。変数は1つの文でまとめて宣言できますし、

```
int a;
int b,c;
```

のように、個別に宣言してもかまいません。このサンプルプログラムの例のように、Cではあらかじめ宣言してからでないと変数を使うことができません。

また変数は型をもっています。その型を変数宣言のときに指定してやります。この例では指定が「int」という型になっています。int型とは、

1 2 10 300 4500

といった小数点以下の値のない数字を格納できる変数です。これに対して、

1.23 10.56

といった数字は実数(浮動小数点数)と呼ばれ、それはintではない別の型名(float。後述)で指定します。

値の代入

宣言した変数に値を代入するには、

```
a=10;
```

のようにします。数学で「a=10」とすると、

――左辺(a)と右辺(10)とが等しい

ということを意味しますが、プログラミング言語の世界では、

――右辺の値を左辺に代入する

という意味になります。つまり機能としてこの式は、

a ← 10;

という雰囲気をもっています。また、

```
a=a+100;
```

とすると、

a ← a+100;

という意味になり、

――aに100を加えたものを新しいaとする

という処理が行なわれます。すなわちaの値を100大きくします。9行目の、

```
c=a+b;
```

も同じ処理で、aとbの和を計算し、その結果を左辺のcに代入します。このとき、aとbの値は元のまま維持されます。

a=10;の意味

a=a+100;の意味

整数型変数の出力

整数値を出力するときは10行目に示したように、

```
printf("%d¥n",c);
```

と記述します。これは整数型変数cの値を出力させるための指示になっています。ここで%dは、

――対応する整数の値を表示しなさい

という指定です。その対応する変数はカンマを1つ入れた後に記述します。

```
        ┌── その後復改しろ
        ↓
printf("%d¥n",c);
        ↑       │
        └───────┘
     cの値を出力しろ
```

出力させる値は変数である必要はありません。たとえば、

```
printf("%d¥n",a+b+100);
```

のように計算式を書くこともできます。

変換文字列

printfの中で用いられている「%d」のことを、

変換文字列

といいます。それは厳密にはこのprintfが、
――対応する値を10進数に変換して出力しろ
という働きをするからです。%dの'd'はdecimal（10進数）を意味します。この他、「16進数に変換して出力しろ」といった役目の変換文字列もあります。代表的な変換文字列を次に示します。

printfの変換文字列

変換文字列	意味
%o	8進数で出力する
%d	10進数で出力する
%x %X	16進数文字で出力する。%xは小文字、%Xは大文字で出力
%f	浮動小数点数として出力する
%c	文字として出力する
%s	文字列として出力する

メッセージを含む数値出力

メッセージを付けて数値出力するときは、

```
例1：printf("aの値=%d¥n",a);
例2：printf("aの値は %d です¥n",a);
```

のようにします。この結果は「aの値=10」、「aの値は 10 です」となります。

複数の変数出力

1つのprintfで複数の変数の値を出力することもできます。たとえば、

```
printf("aの値は%d bの値は%dです¥n",a,b);
```

のようにすると、

```
aの値は10 bの値は20です
```

と表示されます。このとき注意することは、出力したい2つの変数をカンマで区切って並べ、二重引用符の中にも、2つの変数に正確に対応する変換文字列を記述するということです。

```
printf("aの値は%d bの値は%dです¥n",a,b);
```

5.5 実数の値を出力する

リスト　半径から円の面積を計算する

```
 1: #include <stdio.h>
 2:
 3: main()
 4: {
 5:     float pi,r,s;
 6:
 7:     pi=3.14159;
 8:     r=2.0;
 9:     s=pi*r*r;
10:     printf("面積は%fです.¥n",s);
11: }
```

実行結果

```
面積は12.566360です.
```

実数型変数の宣言

5行目は実数型の変数宣言です。

```
float pi,r,s;
```

で「float型の変数、i、r、sをこれから使用する」ということを宣言します。

Cでは、実数型は「浮動小数点型」とも呼ばれます。ただ具体的な型をいうときは、float型(標準精度実数)とかdouble型(倍精度実数)と指定するので、実数型/浮動小数点型の違いは重要ではありません。これは整数型の場合も同じで、具体的な型を指すときはint型(標準整数)、long型(倍長整数)などを使います。

実数型変数の出力

実数型の値を出力するときは、

```
printf("面積は%fです.¥n",s);
```

とします。「%f」が実数値の出力を行なう変換文字列です。'f'はfloatからきています。

5.6 変数の型

Cでよく用いられる基本となる変数型は次のように宣言します。

変数の型宣言

種類	型指定	宣言例
整数	int	int a;
実数	float	float b;
文字	char	char c;
文字列	—	char d[80];

Cでは整数、実数というよく知られた変数型の他に「文字型」があります。これは半角文字1字分の文字コードを格納できる変数です。文字型はcharで示します。charは単独で用いられることもありますが、それを配列表現したものが、文字列用変数としてよく用いられます。たとえば、

```
char d[80];
```

とすると、80文字格納可能な変数であるdを宣言したことになります。Cでは、
――文字列は、文字がつながったもの、つまり文字型変数の配列として表現する
ということになっています。したがってCでは特別に「文字列型」という変数はありません（ただし本書では便宜上、「文字列変数」という言葉を使うことがあります）。

5.7 配列を作る

変数宣言で、

```
int dt[10];
```

とすると、int型の変数を10個用意してくれます。その変数には0からの通し番号が与えられます。0から数えて10個、すなわち、

```
0,1,2,3,4,5,6,7,8,9
```

という通し番号が付けられた変数の集まりdtとなります。ここで、
——3番目のdtを用いたい
というときには、

```
dt[3]
```

という指定をします。つまりこの変数dtは、

```
dt[0],dt[1],dt[2],dt[3],dt[4],dt[5],dt[6],dt[7],dt[8],dt[9]
```

という指定で個別利用できるということです。たとえば、

```
dt[3]=100;
```

とすれば、その変数の値が100になります。このとき[]の中に書く数字のことを「添字」といいます。また、このような変数の集まりのことを、

配列

と呼びます。配列は同じ性質の変数がたくさん必要なときに使います。配列の宣言はどの変数でもできます。たとえば、

```
int a[10];
float b[20];
char c[80];
```

のようになります。すでに説明したように、char型の配列は文字列格納に用いられま

す。配列の使用方法は「9.5　配列にデータを入れる」（→107ページ）で説明します。

int dt[10];による配列変数確保

5.8 例題プログラム

例題 1

int型変数aに5を代入し、aの値、aの2乗値、aの3乗値を表示する

リスト　**解答**

```
#include <stdio.h>

main()
{
    int a;

    a=5;
    printf("a=%d¥n",a);
    printf("aの2乗=%d¥n",a*a);
    printf("aの3乗=%d¥n",a*a*a);
}
```

実行結果

```
a=5
aの2乗=25
aの3乗=125
```

例題2

aの2乗値を変数aaに、aの3乗値を変数aaaに一度格納してから表示する

リスト 解答

```
#include <stdio.h>

main()
{
    int a,aa,aaa;

    a=5;
    aa=a*a;
    aaa=a*a*a;
    printf("a=%d¥n",a);
    printf("aの2乗=%d¥n",aa);
    printf("aの3乗=%d¥n",aaa);
}
```

例題3

float型変数bに2.44949を代入し、その2乗値を変数cに入れてから出力する

リスト 解答

```
#include <stdio.h>

main()
{
    float b,c;

    b=2.44949;
    c=b*b;
    printf("bの2乗=%f¥n",c);
}
```

実行結果

```
bの2乗=6.000001
```

この章のまとめ

- 変数とはデータを記憶しておくものである
- 変数は宣言してから使用する
- 変数には型がある
- 「**a=10;**」は変数**a**に10を入れることを意味する
- 変数の値を出力するときは変換文字列を使用する
- 配列は同じ性格の変数の集まりであり、

```
int st[10];
```

  のように宣言する
- このとき確保される変数は**dt[0]**〜**dt[9]**までである。**dt[10]**はないので注意する
- 文字列用の変数は「**char ss[80];**」のように文字型の配列として宣言する

用例で学ぶ

- 数値出力方法の記述例を以下に示す。

記述	出力
printf("%d¥n",a);	100
printf("%d%d¥n",a,a);	100100
printf("%d %d¥n",a,a);	100 100
printf("data=%d¥n",a);	data=100
printf("加算結果=%d¥n",a+200);	加算結果=300
printf("%f¥n",b);	12.345678
printf("答えは %f です¥n",b);	答えは 12.345678 です

注：次の記述が行なわれているものとする。

```
int a;
float b;
a=100;
b=12.345678;
```

Chapter 6

数字の計算を行なわせてみよう（演算）

6.1 加減乗除を行なう

加減乗除の計算方法は一般のプログラミング言語と同じです。算術対象が整数の場合、割算の結果は小数点以下切り捨てとなりますから注意してください。たとえば、

```
100/3
```

の結果は33となります。

Cプログラムで加減乗除を行なうには、以下の記号を用います。*/は+-より先に計算されます。()があると、その中が先に計算されます。a/b/cは左側から（a/bから）先に計算されます。

演算子（加減乗除）

演算子	機能
+	加算
-	減算
*	乗算
/	除算

記述例

```
n = 2 * 3 + 4 * 5;          ……nは26
n = 16 / 2 - 6 / 2;         ……nは5
n = 2 * ( 3 + 4 ) * 5;      ……nは70
n = 100 / 10 / 5;           ……nは2
```

6.2 剰余を求める

Cでは割算の余りを求めるには%を使います。

演算子（剰余）

演算子	機能
%	剰余

たとえば、

```
100%3
```

の結果は1となります。

除算（/）と剰算（%）

```
a=23/5;
b=23%5;
```

/の答え　%の答え

23 ÷ 5 ＝ 4 余り 3

加減乗除と剰余の計算例を示します。

リスト　100と3の加減乗除を計算する

```
#include <stdio.h>

main()
{
    int a,b,wa,sa,seki,syo,amari;

    a=100;
    b=3;
    wa=a+b;
    sa=a-b;
    seki=a*b;
    syo=a/b;
    amari=a%b;
    printf("和=%d¥n",wa);
    printf("差=%d¥n",sa);
    printf("積=%d¥n",seki);
    printf("商=%d¥n",syo);
    printf("余り=%d¥n",amari);
}
```

実行結果

```
和=103
差=97
積=300
商=33
余り=1
```

6.3 インクリメント演算子とデクリメント演算子

プログラムの中で1加算、1減算という処理は非常によく用いられますので、特別のコンパクト記述ができるようになっています。それは次の通りです。

```
++a; または a++;     →     a=a+1; と同じ
--a; または a--;     →     a=a-1; と同じ
```

ここで++のことをインクリメント演算子（increment：増加）、--のことをデクリメント演算子（decrement：減少）と呼んでいます。

演算子（インクリメント、デクリメント）

演算子	機能
++	1加算
--	1減算

NOTE

++aとa++

インクリメント演算子には2通りの書き方がある。単に**a=a+1;**の代わりに用いるのであれば**++a;**または**a++;**の好きなほうを用いればよい。しかし厳密には両者には次のような違いがある。最初に**a**の値が5だったとする。このとき、

```
b=++a;    ……bの値は6になる。aの値は6になる
b=a++;    ……bの値は5になる。aの値は6になる
```

という結果になる。前者の働きは、

――**a**の値を+1してからその新しい**a**の値を**b**に代入する（全体の演算の前に1加算）

であり、これに対して後者の働きは、
——aの値をbに代入した後でaの値を+1する（全体の演算の後に1加算）
となる。これらを普通の方法で記述すると、

```
b=++a;    →    a=a+1; b=a;
b=a++;    →    b=a; a=a+1;
```

となる。この事情はデクリメント演算子でも同じである。上級者はこの特徴を巧みに用いたプログラミングを行なう。

6.4 例題プログラム

例題1

次の演算結果を確認する

```
8/4/2          ……左から計算される
5*4-20/2+3     ……乗除算が加減算より先に行なわれる
5*4-20/(2+3)   ……かっこ内の計算が最優先される
```

リスト 解答

```
#include <stdio.h>

main()
{
    int a,b,c;

    a=8/4/2;
    b=5*4-20/2+3;
    c=5*4-20/(2+3);

    printf("a=%d b=%d c=%d¥n",a,b,c);
}
```

実行結果

```
a=1 b=13 c=16
```

例題２

インクリメント／デクリメント演算子の働きを確認する

リスト　解答

```
#include <stdio.h>

main()
{
    int a,b;

    a=5;
    ++a;
    printf("a=%d¥n",a);

    b=5;
    --b;
    printf("b=%d¥n",b);
}
```

実行結果

```
a=6
b=4
```

この章のまとめ

計算のための演算子は以下の通りである

演算子	機能	使用例	aの値	コメント
+	加算	a=100+6;	106	
-	減算	a=100-6;	94	
*	乗算	a=100*6;	600	
/	除算	a=100/6;	16	
%	剰余	a=100%6;	4	
++	1加算	++a;	6	最初のaの値を5とする
--	1減算	--a;	4	最初のaの値を5とする

整数除算の場合、小数点以下は切り捨てられるので注意する

Chapter 7

データをキーボードから入力させよう（入力）

print("hello");

7.1 数値を入力し、そのまま出力する

ここではデータ入出力の基本形を覚えることにしましょう。整数、実数、1文字、文字列の入出力をパターンとして知っておけば、いろいろな用途に活用することができます。

次のプログラムはキーボードから入力された数値データを、そのまま画面に出力するものです。

リスト　キーボード入力されたデータを画面出力

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int a;
6:
7:     scanf("%d",&a);
8:     printf("%d¥n",a);
9: }
```

実行結果

```
1234        ……数値を入力した
1234        ……それを画面表示する
```

scanf

数値の入力にはscanfを使います。scanfは変換文字列の指定により、数値だけでなく、文字や文字列の入力もできます。また数値も通常の10進数だけでなく、8進数や16進数、そして実数などの入力もできます。

scanfはprintfとちょうど逆の働きをする関数です。

整数入力の変換文字列

scanfで通常の数値を入力するには、

%d

を使います。これは入力されたデータを10進数と見なして処理するということです。

データ格納変数の指定

このプログラムでは入力されたデータを変数aに格納するように指定しています。それが、

```
scanf("%d",&a);
```

の部分です。ここで変数aに入力するのなら、

```
scanf("%d",a);          ……'&'がない
```

とすべきではないか、という気がします。これはきわめて自然な疑問ですが、そうするわけにはいきません。それはscanf関数が、

——データの格納先は変数名ではなく、その変数の置かれているアドレスで指定しなければならない

という仕様になっているからです。それに合わせると&aとなるわけです。しかし初心者は細かい理屈は考えずに、

scanfで数値変数にデータ格納をするときは&を付加する

ということをまず覚えてください。

scanf("%d",&a);の働き

1234
変数a
メモリ
10進数入力(%d)
1234を入力
1 2 3 4
キーボード

7.2 実数の入出力を行なう

実数入力を行なうときは「その数字が実数である」という指示が必要になります。実数入力を行なうプログラムを次に示します。

リスト　入力された半径から円周と円面積を計算する

```
 1: #include <stdio.h>
 2:
 3: main()
 4: {
 5:     float pai=3.14159;
 6:     float r,l,s;
 7:
 8:     printf("半径を入力してください:");
 9:     scanf("%f",&r);              ……実数の数値を変数rに入力する
10:     l=2*pai*r;
11:     s=pai*r*r;
12:     printf("円周は%fです.¥n",l);
13:     printf("面積は%fです.¥n",s);
14: }
```

実行結果

```
半径を入力してください:5.2
円周は32.672535です.
面積は84.948593です.
```

変数の初期化

5行目の変数宣言である、

```
float pai=3.14159;
```

はfloat型変数paiを宣言し、同時にその値を3.14159にすることを意味しています。これは機能としては、

```
float pai;
pai=3.14159;
```

とするのと同じことです。このような機能を初期化といいます。同じようにint型変数の初期化は、

```
int a=100;
```

のようにします。

実数入力の変換文字列

scanfで通常の実数を入力するには、

%f

という変換文字列を使います。これにより入力されたデータを実数と見なして処理してくれます。それ以外の記述は整数入力の場合と同じです。実数格納用変数を指定する場合も変数名に'&'を付加します。

scanf("%d",&r);の働き

5.2
変数r
メモリ
実数入力(%f)
5.2を入力
5 . 2
キーボード

入力促進メッセージの出力

数値入力をするときは、ユーザーを迷わせないために、入力促進メッセージを出力したいものです。

しかし、Cではあいにくscanfは入力専用の機能しかないため、

```
scanf("半径を入力してください:%f",&r);          ……誤り
```

などとすることはできません。入力促進メッセージを出力するためには、

――scanfに先だってprintfでメッセージ出力する

という方法をとります。たとえば、

```
printf("半径を入力してください:");
scanf("%f",&r);
```

とします。

7.3 1文字入力し、そのまま出力する

Cでは1文字単位でデータの入出力を行なうための方法が用意されています。これを使いこなせるようになると大変便利です。次のプログラムは、キーボードから入力された1文字を、そのまま画面に出力するものです。

リスト 入力された文字をそのまま出力する

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     int ch;                      ……char型でなくint型
6:
7:     ch=getchar();
8:     putchar(ch);
9: }
```

実行結果

```
A  ……1文字を入力した
A  ……それを画面表示する
```

getchar

1文字入力を行なうための関数がgetcharです。

```
ch=getchar();
```

と記述することで、キーボードから入力された1文字が変数chに格納されます。なお、このとき使用する変数chには文字（正確には文字コード）が入力されますが、その変数型としてはchar型ではなく、さまざまな安全策を考慮してint型を使用するのが標準手法となっています（これには理由があるのですが、ここでは述べません）。

putchar

putcharはgetcharと対をなす関数で、1文字を画面に出力します。

```
putchar(ch);
```

と記述することで、変数chに格納されている1文字を画面に表示します。

なお変数chの代わりに、文字を直接指定することもできます。また文字コードを直接指定してもかまいません。次に例を示します。

```
putchar('C');     ……'C'と表示
putchar('3');     ……'3'と表示
putchar(65);      ……'A'と表示
```

NOTE

文字コード

コンピュータは文字を、文字コード（キャラクタコード）と呼ぶ連続番号で管理している。基本となるASCII（アスキー）文字コード表は0から127までの番号からなる。たとえば、

```
ch='A';
```

とすると変数chには英文字'A'に相当する文字コードである65が代入される。また、

```
ch=getchar();
```

のとき、'A'という文字をキーボードから入力すると、変数chには65が入る。

日本においては、このコード表を拡張したJIS文字コード表が使われている。これはASCII文字コード表を一部手直しし、カタカナコードなどを追加したものである。ASCII文字コードのうち、0から31までの特殊コードを除外した英数字・記号は次のコードになっている。

主要文字コード一覧

10進	16進	文字
32	20	空白
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

10進	16進	文字
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	¥
93	5D	]
94	5E	^
95	5F	_

10進	16進	文字
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

getcharとputcharの働き

文字Aをキーイン

ch=getchar();

A

キーボード

65

65が入る　変数ch

文字Aが表示される

putchar(ch);

A

ディスプレイ

65

65が入っている　変数ch

7.4 文字列を１行入力し、そのまま出力する

文字列を入出力するためには、gets、putsを使います。次のプログラムは文字列入出力のもっとも簡単な例です。

リスト　入力された文字列をそのまま表示する

```
1: #include <stdio.h>
2:
3: main()
4: {
5:      char s[80];
6:
7:      gets(s);                    ……文字列をsに入力
8:      puts(s);                    ……文字列sを出力
9: }
```

実行結果

```
abc123xyz        ……入力した文字列
abc123xyz        ……それを出力
```

文字列格納用変数

5行目の、

```
char s[80];
```

は文字列を格納するための変数（以下、文字列変数と呼ぶ）です。この文字列変数には最大80文字まで格納することができます。それより長い文字列が格納されると、オーバーした分がメモリ内の隣接部分にむりやり格納され、そこに前からあった内容を破壊します。これはときには暴走の原因になります。したがってプログラマは最大何文字の文字列入力がありうるかを考慮し、それに対して十分なサイズの文字列変数を用意しなければなりません。もし40文字必要なら、ちょっと余裕をみて、

```
char s[50];
```

にするといった配慮が必要です。

文字列終端マーク

文字列を変数に入れた場合、
――どこまでが文字列か
という判断が必要となります。Cではそのために文字列終端マークを用意しています。それは数値の0です。たとえば"ABC"という3文字からなる文字列を文字列変数sに格納したとき、文字列データは、

	s[0]	s[1]	s[2]	s[3]	s[4]	·····
s	65	66	67	0		

という形で格納されます（65は'A'の文字コード）。このため、3文字の文字列を格納するには最低でも4文字分の文字列変数を必要とします。逆にいうと、

```
char s[80];
```

という変数に実際に格納できる文字列の長さは79文字までということになります。なお、この例ではs[0]〜s[3]までが意味のあるデータで、s[4]以降にはどんなデータがあっても無視されます。

gets

キーボードから文字列を入力するためにはgetsを使います。

```
gets(s);
```

とすることにより、変数sに文字列が格納されます。

puts

文字列変数に格納されている変数を出力するためにはputsを使います。

```
puts(s);
```

とすることで、変数sに格納されている文字列が出力されます。putsはprintfによる文字列出力と違って、

——出力した後、自動的に復改する

という処理を行ないます。またputsでも文字列を直接記述することができます。たとえば、

```
puts("test program");
```

とすると、該当する文字列を出力した後に復改します。putsはprintfと違って「文字列だけ」を出力する関数です。printfは文字列も数字も出力できます。

gets と puts の働き

7.5 例題プログラム

例題1

入力した文字列をputsとprintfでそれぞれ2回ずつ出力する

リスト 解答

```
/* smpp9.c */
#include <stdio.h>

main()
{
    char buf[80];

    printf("string=");
    gets(buf);
    puts(buf);
    puts(buf);
    printf("%s¥n",buf);
    printf("%s¥n",buf);
}
```

実行結果

```
string=C language
C language
C language
C language
C language
```

例題2

入力された秒数を、時間、分、秒に変換する

リスト **解答**

```
/* smpp10.c */
#include <stdio.h>

main()
{
    int ji,fun,byou;

    printf("秒数=");
    scanf("%d",&byou);
    ji=byou/3600;
    byou=byou%3600;
    fun=byou/60;
    byou=byou%60;

    printf("%d時間 %d分 %d秒¥n",ji,fun,byou);
}
```

実行結果

```
秒数=3678
1時間 1分 18秒
```

この章のまとめ

- 文字列には終端マークとして0が付加される
- データ入出力の基本形は次の通りである

データ種別	入力方法	出力方法
1文字	a=getchar();	putchar(a);
文字列	gets(s);	puts(s);
整数値	scanf("%d",&a);	printf("%d¥n",a);
実数値	scanf("%f",&b);	printf("%f¥n",b);

注：以下の変数宣言が行なわれているものとする。

```
int   a;
float b;
char  s[80];
```

- **printf** は文字列も数値も出力できるが、**puts** は文字列だけを出力する。また **puts** は自動的に復改処理をする

用例で学ぶ

- printf と puts による文字列出力の比較例を以下に示す。文字列変数 str の内容は "abcde"であるとする。

記述	出力
puts(str);	abcde
printf("%s¥n",str);	abcde
puts("MYSTRING");	MYSTRING
puts("MY¥nSTRING");	MY STRING
printf("MYSTRING¥n");	MYSTRING
puts(str); puts(str);	abcde abcde
printf("%s",str); printf("%s¥n",str);	abcdeabcde
printf("文字列は[%s]¥n",str);	文字列は[abcde]

Chapter 8
ある条件のときだけ処理させよう（条件文）

8.1 日常生活での条件判断

私たちは日常生活の中で、常に条件判断をしています。それらはたいてい本能的に行なわれています。たとえば、

——雨なら傘をさす

という行動をとります。これは自然にそのような行動になってしまいますが、行動原理からいえば立派な条件判断です。この行動は機能としては、

——もし雨なら、傘をさせ

という条件判断になっています。人間の体の働きもさまざまな条件判断で成り立っています。

- 目にゴミが入ったら、涙で押しだせ
- 暑ければ、汗をだせ
- 何かが飛んできたら、とっさに身をすくめろ
- 障害物があったら、止まれ

これらも皆、条件判断です。もし適切な条件判断がなければ、とんでもないことになります。たとえば小学校の先生が生徒に、

——まっすぐ家に帰るのですよ

といっても、元気のいい大将が必ずいて、

——まっすぐ歩いたら壁にぶつかりまーす

とバカをいいます。これも実は言外に「適切に条件判断して」という意味をもっているのは当然です。

この条件判断は、プログラムでももっとも重要なものです。条件判断ができないと、プログラムの機能は限定されてしまいます。だからプログラミング言語には必ず、条件文が用意されています。それが`if`文です。`if`文では次のような条件判断を行ないます。

条件判断　その1

もし○○なら　　△△を実行せよ

条件判断　その2

もし○○なら　　△△を実行せよ
さもなくば　　　××を実行せよ

この両者は実は同じことです。「条件判断　その2」において、「さもなくば」以降が不要なら、これを省略することが可能で、それが「条件判断　その1」となります。

8.2 if文を使ったプログラム

Cでは条件判断はif文が行ないます。次のプログラムは、

定価が1000円以上なら2割引する
そうでないときは1割引する

という処理をします。

リスト　商品の実売価格を計算する

```
 1: /* iftst.c */
 2: #include<stdio.h>
 3: main()
 4: {
 5:     int kane;
 6:
 7:     printf("定価:");
 8:     scanf("%d",&kane);
 9:
10:     if(kane>=1000){          ……1000円以上なら
11:         kane=kane*0.8;       ……2割引
12:     }
13:     else{                    ……さもなくば
14:         kane=kane*0.9;       ……1割引
15:     }
16:
17:     printf("価格=%d¥n",kane);
18: }
```

実行結果

```
C>iftst
定価:500        ……1000円未満なら
価格=450        ……1割引

C>iftst
定価:2000       ……1000円以上なら
価格=1600       ……2割引
```

if文

10〜15行目が条件判断をするif文です。これがもっとも基本的なif文の用例です。このif文は、

──条件によってAの処理をするかBの処理をするかを決める

というときに用いられます。このif文の機能は次の通りです。

if-else型条件文

```
if(条件){
   条件がとれたときに実行する文1
}
else{
   条件がとれないとき実行する文2
}
```

真　条件　偽

文1　文2

条件判断の例

```
if(お金があれば){   ← この条件がとれていたら
   すぐにパソコンを買う
}
else{   ← そうでないときは
   次のボーナスまで待つ
}
```

賞与

ブロック文

if文やfor文で制御される文はいくつあってもかまいません。そのときどこからどこまでが制御される範囲なのか、を示すためにブロック文を使います。ブロック文は{}で示します。その意味は、

```
{                  ……ここから
}                  ……ここまで
```

ということです。if文の条件が満たされていると、最初のブロック文の範囲内に書かれた文を実行します。条件が満たされていないと、else句に続くブロック文の範囲内に書かれた文を実行します。実行される文がいくつもあるときは、

```
if(a>10){          ……aが10より大なら
    b=1;           ……bを1にして
    c=2;           ……cを2にする
}
else{              ……さもなくば
    b=3;           ……bを3にして
    c=4;           ……cを4にする
}
```

のように、{から対応する}までの間に書き並べます。

ブロック文の省略

実行する文が1つだけの場合は、特別にブロック文を省略することができます。たとえば10～15行目のプログラムは、それぞれ実行文が1つしかないので、

```
if(kane>=1000)
    kane=kane*0.8;
else
    kane=kane*0.9;
```

と書くこともできます。

else句の省略

if文の基本形はこの例で示す、

if-else型

です。しかし、else句以下が不要ならそれは省略できます。たとえばもし
――定価が1000円以上なら2割引する
というだけのif文なら、

```
if(kane>=1000){          ……1000円以上なら
    kane=kane*0.8;       ……2割引
}
```

となります。このときif文の機能は次のようになります。

elseのない型条件文

```
if(条件){
  条件がとれたときに実行する文
}
```

条件 → 真 → 文1 / 偽

関係演算子

if文の条件は「条件式」で記述します。その条件式は「関係演算子」という演算子を使って記述します。
関係演算子は、
――2つの値の大小関係や等値関係を判定するもの
です。関係演算子をまとめると次のようになります。

関係演算子

演算子	説明	記述例	記述の意味
>	大きい	if(a>100)	aが100より大なら
>=	大きいか等しい	if(a>=100)	aが100以上なら
<	小さい	if(a<100)	aが100より小なら
<=	小さいか等しい	if(a<=100)	aが100以下なら
==	等しい	if(a==100)	aが100なら
!=	等しくない	if(a!=100)	aが100でないなら

真と偽

条件式の判定結果は「条件が正しくとれていた」場合のことを「真」と表現します。また「条件が正しくとれていなかった」場合のことを「偽」と表現します。たとえば変数aの値が100であったとき、

if(a>50)　　……条件式の結果は真
if(a>200)　　……条件式の結果は偽

と表現します。この真、偽は通称であり、実際には真は1(厳密には0以外なら真)、偽は0という値をもちます。

真偽による判断

```
a=100;
if(a>50){      ┐ 真なので次の文を実行する
    b=0;   ←   ┘
}

a=20;
if(a>50){      ┐
    c=0;       │ 偽なのでスキップする
}              │
←──────────────┘
```

8.3 論理演算子で込み入った条件を記述する

通常、条件式には1つの式を書きますが、場合によっては2つ以上の条件式を書く必要のあるときがあります。たとえば、

「**a>10**」かつ「**a<100**」なら
「**a==100**」または「**a==200**」なら

といったものです。この中の「かつ」、「または」に相当するものが論理演算子と呼ばれるものです。
次のプログラムはうるう年判定を行なうものです。論理演算子を使っています。

リスト　西暦から「うるう年」かどうか判定する

```
 1: /*  uruu.c   */
 2: #include <stdio.h>
 3: main()
 4: {
 5:     int n;
 6:
 7:     printf("年=");
 8:     scanf("%d",&n);
 9:
10:     if(n%4==0 && n%100!=0){
11:         printf("うるう年です¥n");
12:     }
13:     if(n%400==0){
14:         printf("うるう年です¥n");
15:     }
16: }
```

実行結果

```
C>uruu
年=1992
うるう年です

C>uruu
年=1900

C>uruu
年=2000
うるう年です
```

「かつ」の表現

うるう年は、

　4で割り切れ　かつ　100で割り切れない年
　400で割り切れる年

のどちらかに当てはまる年のことです。このうるう年を判断しているのが10行目と13行目のif文です。最初の条件は10行目で次の剰余計算により判定されます。

```
                    ┌──────────── 年を4で割った余りが0であり
                    │     ┌────── かつ
                    │     │    ┌─ 年を100で割った余りが0でない
                    ↓     ↓    ↓
    if (n%4==0 && n%100!=0)
```

この記述は詰め気味に書かれていますが、もっと読みやすくするために空白を入れて、

```
if( n%4 == 0 && n%100 != 0)
if( n % 4 == 0 && n % 100 != 0)
```

などとする人もいます。これはプログラマの好みの問題です。ここで、

```
&&
```

が「かつ」に相当する論理演算子で、「アンド」と呼ぶこともあります。

「または」の表現

このサンプルプログラムはアンド（かつ）の表現でしたが、もう1つ「または」の表現も重要です。たとえば、

——aの値が10未満または100より大なら

という条件は、

```
if(a<10 || a>100)
```

となります。ここで、

```
||
```

が「または」に相当する論理演算子で、「オア」と呼ぶこともあります。

先の「うるう年判断」は2つのif文からなっていましたが、これに「または」の演算子を入れると、

```
if((n%4==0 && n%100!=0) || n%400==0){
    printf("うるう年です¥n");
}
```

と1つの文で記述することができます。これは、

「4で割り切れ かつ 100で割り切れない年」 または 「400で割り切れる年」

という意味になります。ご覧のようにプログラムは短くなりますが、それだけ複雑な条件になります。通常は複雑な条件式は分割して、なるべく単純にするのがよいプログラムです。

「でない」の表現

論理演算子には「でない」という否定表現の機能もあります。これは「ノット」と呼ばれることもあります。たとえば、

——aの値が1000以上、でないなら

という条件は、

```
if(!(a>=1000))
```

となります。ここで、

```
!
```

が「でない」に相当する論理演算子です。この条件式で、内側にあるかっこは「aの値が1000」という条件全体を否定するために必要です。また、

——bの値が10または100以上、でないなら

という条件は、

```
if(!(b==10 || b>=100))
```

となります。

論理演算子の整理

論理演算子を整理すると次のようになります。

論理演算子

演算子	説明	記述例	記述の意味
&&	論理積（かつ／and）	if(a==3 && b==4)	aが3かつbが4なら
||	論理和（または／or）	if(a==3 || b==4)	aが3またはbが4なら
!	否定（でない／not）	if(!(a==3 && b==4))	aが3かつbが4、でないなら

関係演算子･論理演算子の真偽値判断

表の見方 ○：条件判断の結果が真
×：条件判断の結果が偽

```
if(a==3 && b==4)
```

		bの値	
		4	4以外
aの値	3	○	×
	3以外	×	×

```
if(a==3 || b==4)
```

		bの値	
		4	4以外
aの値	3	○	○
	3以外	○	×

```
if(!(a==3 && b==4))
```

		bの値	
		4	4以外
aの値	3	×	○
	3以外	○	○

```
if(!(a==3 || b==4))
```

		bの値	
		4	4以外
aの値	3	×	×
	3以外	×	○

優先順位の指定

条件式にはかっこを用いることがあります。このかっこは、その中の判断を先に行なわせるように優先順位を指示します。これはときには、式を見やすくするためにも用いられます。優先順位設定の例を示します。

式：　`if((a==0 || b==0) && c==0)`
意味：　aが0またはbが0で、かつ、cが0なら

式：　`if(a==0 || (b==0 && c==0))`

意味：　aが0、または、bが0かつcが0なら

8.4 例題プログラム

例題1

入力された数字が奇数か偶数かを判別する

リスト　解答

```
/* iftst.c */
#include <stdio.h>

main()
{
    int n;

    printf("n=");
    scanf("%d",&n);
    if(n%2 == 0){
        printf("偶数です¥n");
    }
    else{
        printf("奇数です¥n");
    }
}
```

実行結果

```
C>iftst
n=123
奇数です

C>iftst
n=126
偶数です
```

例題2

入力された1文字が16進数を構成する文字であるかどうかを判定する。16進数構成文

字は、

0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

からなる。ここでは文字コードの連続性に着目し、

'**0**'以上 かつ '**9**'以下	('0'<=c && c<='9')
または	||
'**A**'以上 かつ '**F**'以下	('A'<=c && c<='F')
または	||
'**a**'以上 かつ '**f**'以下	('a'<=c && c<='f')

という条件判断を行なっている。

注：この目的のためには**isxdigit**という関数が用意されている(後述する)。

リスト 解答

```
/*  hex.c  */
#include <stdio.h>

main()
{
    int c;

    printf("文字=");
    c=getchar();

    if(('0'<=c && c<='9') || ('A'<=c && c<='F')
                          || ('a'<=c && c<='f')){
        printf("16進数文字です¥n");
    }
    else{
        printf("16進数文字ではありません¥n");
    }
}
```

実行結果

```
C>hex
文字=A
16進数文字です
```

```
C>hex
文字=g
16進数文字ではありません
```

この章のまとめ

- 条件判断は`if`文で行なう
- 実行される文はブロック文`{}`の中に書く
- 実行される文が１つの場合はブロック文`{}`を用いなくてもよい
- `if`文の書式は、

  `if(条件) {...} else {...}`

  であり、`else`句が不要なら、

  `if(条件) {...}`

  となる
- 関係演算子には、`>`、`>=`、`<`、`<=`、`==`、`!=`がある
- 複数の算術演算を並べるときは論理演算子を用いる
- 論理演算子には、`&&`（and）、||（or）、`!`（not）がある

用例で学ぶ

- `if`文の記述例を次に示す。

記述	説明
`if(a==0)` `    b=1;`	Cの制御文は単文を制御する

`if(a==0){`
`    b=1;`
`    c=2;`
`}`
	複数の文を記述するときは`{}`が必要

`if(a==0){`
`    b=1;`
`}`
	単文のときは、あえて`{}`を用いてもよい

`if(a>10){`
`    b=0;`
`    c=1;`
`}`
`else{`
`    b=2;`
`    c=3;`
`}`
	if-else型の例

`if(a>10) b=1; else b=2;`	短いif文はこのように書いてもよい

Chapter 9

何度も繰り返して処理させよう（繰り返し文）

LOOP...

9.1 繰り返し処理の必要性

私たちは日常、同じことを何度も繰り返しながら生活しています。たとえば、
──「1日に3度食事すること」を毎日繰り返す
ということをしています。また学生は、
──「1週間分の時間割表」通りの行動を毎週繰り返す
という生活をしています。ここでは、

```
月曜　国語　理科　社会　数学
火曜　英語　技術　体育　音楽
　　　　　　：
　　　　　　：
土曜　美術　国語　英語　社会
```

といった繰り返しの基本パターンが表示されています。もしこの「繰り返す」という了解がなかったら、時間割表は、

```
 4月　1日　国語　理科　社会　数学
 4月　2日　英語　技術　体育　音楽
　　　　　　：
12月15日　美術　国語　英語　社会
　　　　　　：
```

のように、延々と1年間分の表割りを作らなければならなくなります。繰り返しを用いると、ものごとをコンパクトに表現できるという利点があります。

この事情はプログラムでも同じです。たとえば画面に"Hello"と10回表示させるためには、

```
printf("Hello¥n");
printf("Hello¥n");
printf("Hello¥n");      printfが10回
  :
  :
printf("Hello¥n");
```

とすれば実現できます。しかし、これはすぐに察っせられるように、いかにも間の抜けた方法です。500回繰り返せ、となったら途方にくれてしまいます。この処理は概念的に、

次の文を10回繰り返せ

```
    printf("Hello¥n");
```

とするほうがスマートです。このような処理を行なうために繰り返し文が用意されています。プログラミング言語の世界では、繰り返しのことをループ(loop：輪)といいます。たとえば「ループ処理する」とか「無限ループになった」というように使います。

日常生活の繰り返し処理

●人生80年とは…

夏　秋　冬　春

春夏秋冬を80回繰り返すことである

●学生は…

水　木　金　土　日　月　火

	月	火	水	木	金	土
1	国	音	社	理	国	算
2	算	算	国	算	社	国
3	理	体	算	国	体	
4	社	理	体	図	理	

1週間の時間割表を卒業まで繰り返す

●1日は…

6　12　18　24　1日　92　183　275　365　1年

「1時間」を24回繰り返すものであり、
1年は「1日」を365回繰り返すものである

●意志の弱いスモーカーは…

もうやめた　やっぱり吸おう

…を何回も繰り返す

9.2 for文を使ったプログラム

Cで繰り返し処理をする代表的な文はforです。次のプログラムは、

画面に"**Hello**"と10回表示させる

という働きをします。

リスト　画面に「Hello」と10回表示させる

```
1: #include <stdio.h>
2: main()
3: {
4:      int i;
5:
6:      for(i=1; i<=10; i++){              ……10回繰り返せ
7:          printf("Hello¥n");             ……この文を
8:      }
9: }
```

実行結果

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

for文

6～8行目が繰り返し処理をするfor文です。これがもっとも基本的なfor文の用例です。このfor文は目的としては、

――後続するブロックの中に書かれた文を10回繰り返せ

という機能になっています。その「10回繰り返せ」という目的を具体化するために、変

数iを使い、
——変数iの値が1から10になるまで繰り返せ
という記述をしています。このfor文の記述は次の意味をもちます。

最初に変数iの値を1にしろ
iの値が10以下の間繰り返せ
毎回の処理のたびにiの値を1増やせ
この'{'から

```
for(i=1; i<=10; i++){
    printf("Hello¥n");
}
```

繰り返し実行文の集まり
この'}'までに書かれた文を実行しろ

for(i=1; i<=10; i++)の働き

for(i=1; i<=10; i++)

はじめ
iの値を1にする
iの値が10以下か？
YES
NO
"Hello"と表示する
iの値を1増やす
おわり

for文の決まり文句

Cのfor文は多彩な機能をもっていますが、もっとも基本となる用法はここに示した、
——変数○の値が△から×になるまで繰り返す

というものです。さらにこのときの変数にはiがよく使われます。iはiteration(繰り返し)の先頭文字です。そして数え上げは「1から」行なうのが普通です。これらを考慮するとfor文の決まり文句は次のようになります。

for文の決まり文句

```
for(i=1; i<=回数; i++){

    実行したい文の集まり

}
```

「回数」のところには数値を直接書くこともできますし、変数で指定することもできます。たとえば次のようになります。

例1：200回繰り返せ

```
for(i=1; i<=200; i++){
    ...
}
```

例2：変数**n**の値だけ繰り返せ

```
for(i=1; i<=n; i++){
    ...
}
```

例3："**i=i+1**"を使う

```
for(i=1; i<=n; i=i+1){
                        /* i++ の代わりにこのように普通に書いてもよい */
    ...
}
```

例4：2つずつ数が増えるようにしろ

```
for(i=1; i<=n; i=i+2){  /* i=i+2 にする */
    ...
}
```

9.3 制御用変数を利用する

for文で用いられる制御用変数は、ただループ回数を管理するだけでなく、その値を直接使ってさまざまな演算を行なうことができます。次のプログラムは、

1から5までの和と積を計算する

というものです。

リスト　1から5までの和と積を計算する

```
 1: #include <stdio.h>
 2: main()
 3: {
 4:     int i,wa,seki;
 5:
 6:     wa=0;
 7:     seki=1;
 8:     for(i=1; i<=5; i++){
 9:         wa=wa+i;
10:         seki=seki*i;
11:     }
12:     printf("和=%d¥n",wa);
13:     printf("積=%d¥n",seki);
14: }
```

実行結果

```
和=15
積=120
```

制御用変数の値を利用する

8行目のfor文では決まり文句通り、変数iを利用しています。この変数iの値は、ループ回数を数えるだけでなく、その実行文の中で自由に利用することができます。ここでは、9、10行目で、

```
wa=wa+i;
seki=seki*i;
```

のように変数iの値を使用しています。

9.4 while文を使ったプログラム

前項で説明したのは「10回繰り返す」のように、
——繰り返す回数が最初からわかっている
という場合の繰り返し方法です。それを行なうのがfor文でした。
ループ処理では、この他にもう1つ、
——繰り返す回数が最初はわからない
という処理があります。たとえば次の問題を考えてみましょう。

簡易電卓プログラム

・キーボードから数字を入力していく
・その数字を加算していく
・今扱った数値が0なら処理を終了して合計値を表示する

この場合、繰り返し処理が必要ですが、いつその繰り返しを終了するかは、入力されるデータ次第です。ここでは、
——0が入力されたら終了
ということになります。つまり「繰り返す回数が最初はわからない」わけです。while文で対応したプログラムを次に示します。

リスト 入力された数字の合計を計算する

```
 1: /* sum.c */
 2: #include <stdio.h>
 3: main()
 4: {
 5:      int dt=-1,sum=0;
 6:
 7:      while(dt != 0){
 8:          printf("? ");
 9:          scanf("%d",&dt);
10:          sum=sum+dt;
11:      }
12:      printf("合計=%d¥n",sum);
13: }
```

実行結果

```
C>sum
? 100
? 200       入力した数値を加算する
? -50
? 0         ……0の入力でループ処理を終了する
合計=250    ……合計値を表示する
```

while文

7〜11行目が「繰り返す回数が最初はわからない」ときに用いるループであるwhile文です。while文ではループのたびごとにループ条件を見て、それが正しければループ処理を続行します。条件が正しくなければ、ループ処理を打ち切ります。

このwhile文の記述は次の意味をもちます。

```
           ┌─ この条件がとれている間
while( dt != 0 ){ ── この'{'から

   printf("? ");
   scanf("%d",&dt);   繰り返し実行文の集まり
   sum=sum+dt;

} ─────────── この'}'までに書かれた文を実行しろ
```

ここでループ条件は、

```
dt != 0        ……dtが0でない間ループしろ
```

ということになっています。これは別の表現をすれば、

──dtが0になったらループ終了

を意味します。また、このループ条件のポイントとなる変数dtの値は、ループ範囲の中の実行文のどれかで、

──毎回のループごとにその値が更新される

という形に記述されていなければなりません。このプログラムではループごとに、

```
scanf("%d",&dt);
```

を実行して、変数dtに新しい数値を読み込んでいます。この変数dtの値の塗り替えを行なわずに、

```
while(dt != 0){

    printf("? ");
    sum=sum+dt;

}
```

変数dtの値が変化しない

といったプログラムを書くと、終了条件が永久にとれなくなり、無限ループになってしまいます。

while(dt != 0)の働き

whileループの前処理

5行目は変数宣言とともに、これから行なうwhileループのための前処理を行なっています。ここでは初期化で値設定を行なってみました。まずwhileループの条件が、

——dtの値が0でないこと

ですから変数dtの値を−1に仮に設定し、とりあえず最初のループ条件に合致するようにしています。変数dtの値は最初のscanfで変更されるので、合計値には影響を与

えません。また合計値を正しくとるために変数sumの値を最初に0にクリアしています。

whileループの決まり文句

while文を使うときにも決まり文句(標準パターン)があります。それは次の通りです。

```
最初の条件要素設定

while( 繰り返し条件 ){

    実行したい文の集まり
    繰り返し条件要素の再設定

}
```

while文の決まり文句を利用した簡単な例をあげてみます。for文の、

```
for(i=1; i<=10; i++){
    実行文の集まり
}
```

と同じ働きをするプログラムをwhile文で作ると次のようになります。

```
i=1;                   ……最初の条件要素設定
while(i <= 10){        ……繰り返し条件
    実行文の集まり      ……繰り返し実行文をここに書く
    i=i+1;             ……条件要素の再設定
}
```

9.5 配列にデータを入れる

次のプログラムは0から19までの数字について、その2乗値と3乗値を計算し、一度配列変数に格納したうえで画面表示させるものです。

リスト　0〜19までの2乗値と3乗値を計算する

```
 1: #include <stdio.h>
 2: main()
 3: {
 4:     int i;
 5:     int sqr[20],cube[20];                  ……配列の宣言
 6: 
 7:     for(i=0; i<=19; i++){
 8:         sqr[i]=i*i;                        ……2乗値の設定
 9:         cube[i]=i*i*i;                     ……3乗値の設定
10:     }
11: 
12:     printf("数  2乗  3乗¥n");
13: 
14:     for(i=0; i<=19; i++){
15:         printf("%2d %4d %4d¥n",i,sqr[i],cube[i]); ……配列内容の表示
16:     }
17: }
```

実行結果

```
数 2乗 3乗
 0    0    0
 1    1    1
 2    4    8
 3    9   27
 4   16   64
 5   25  125
 6   36  216
 7   49  343
 8   64  512
 9   81  729
10  100 1000
11  121 1331
12  144 1728
13  169 2197
14  196 2744
15  225 3375
16  256 4096
17  289 4913
18  324 5832
19  361 6859
```

配列の宣言

5行目の、

```
int sqr[20],cube[20];
```

は2乗値を入れる変数の集まりsqrと、3乗値を入れる変数の集まりcubeを宣言しています。この配列はfor文を使って効果的に処理していきます。

配列へのデータの設定

配列にデータを設定するときは添字をうまく活用してやります。つまり変数を、

sqr[i]

と表現し、変数iの値を0～19と変更してやれば、sqr[0]～sqr[19]を指定したことになります。実際にそのような指定をしているのが7～10行目の、

```
for(i=0; i<=19; i++){
    sqr[i]=i*i;
    cube[i]=i*i*i;
}
```

の部分です。ここではそれぞれの配列に、基本となる数字iの2乗値と3乗値を格納しています。また14～16行目の、

```
for(i=0; i<=19; i++){
    printf("%2d %4d %4d¥n",i,sqr[i],cube[i]);
}
```

という記述は、同じ方法で配列の内容を表示しています。

数字の表示幅指定

数字を表示させるとき、桁数を揃えたいことがあります。たとえば、

1
23
337
5262

ではなく、

```
   1
  23
 337
5262
```

としたいときがあります。この指示はC言語では、
——数字を4文字幅で表示しろ
と指定することで可能です。それを行なっているのが15行目のprintfです。これは次の意味をもっています。

```
printf("%2d %4d %4d¥n",i,sqr[i],cube[i]);
```

iの値を2文字幅10進数で表示しろ
cube[i]の値を4文字幅10進数で表示しろ
sqr[i]の値を4文字幅10進数で表示しろ

9.6 例題プログラム

例題1:文字コード一覧表を出力する その1

文字と文字コードとの対応表を作る。printf中の%cは、対応する変数iの値を文字コードと見なし、それに応じた1文字を出力させるための変換指示。たとえばiの値が65のときは対応する文字'A'を出力する。

リスト 解答

```
#include <stdio.h>

main()
{
    int i;
```

```
    printf("文字 文字コード¥n");
    for(i=32; i<=126; i++){
        printf(" %c       %d¥n",i,i);   ……iの値を文字と10進数で出力
    }
}
```

実行結果

```
文字 文字コード
          32       ……これは空白文字
 !        33
 "        34
 #        35
 $        36
 %        37
 &        38
 '        39
 (        40
 )        41
 *        42
 +        43
 ,        44
 -        45
 .        46
 /        47
 0        48
 1        49
 2        50
 3        51
     :
（途中省略）
     :
 z        122
 {        123
 |        124
 }        125
 ~        126
```

例題2：文字コード一覧表を出力する　その2

文字と文字コードとの対応表を作る。1行で8文字分ずつ出力させるように整形する。文字コードが8で割り切れるときに改行を行なうことで表形式にすることができる。文字コードは16進数で表示する。
printf中の%Xは、対応する変数iの値を16進数で出力させるための変換指示。

リスト　解答

```
#include <stdio.h>

main()
{
    int i;

    for(i=32; i<=126; i++){
        if(i%8 == 0){                          ……8で割って余りが0なら
            printf("¥n");                      ……改行する
        }
        printf("%c %X     ",i,i);              ……iの値を文字と16進数で出力
    }
}
```

実行結果

```
  20     ! 21     " 22     # 23     $ 24     % 25     & 26     ' 27
( 28     ) 29     * 2A     + 2B     , 2C     - 2D     . 2E     / 2F
0 30     1 31     2 32     3 33     4 34     5 35     6 36     7 37
8 38     9 39     : 3A     ; 3B     < 3C     = 3D     > 3E     ? 3F
@ 40     A 41     B 42     C 43     D 44     E 45     F 46     G 47
H 48     I 49     J 4A     K 4B     L 4C     M 4D     N 4E     O 4F
P 50     Q 51     R 52     S 53     T 54     U 55     V 56     W 57
X 58     Y 59     Z 5A     [ 5B     ¥ 5C     ] 5D     ^ 5E     _ 5F
` 60     a 61     b 62     c 63     d 64     e 65     f 66     g 67
h 68     i 69     j 6A     k 6B     l 6C     m 6D     n 6E     o 6F
p 70     q 71     r 72     s 73     t 74     u 75     v 76     w 77
x 78     y 79     z 7A     { 7B     | 7C     } 7D     ~ 7E
```

例題3：10000を越えない最大の倍数を見つける

数字を入力する。その数字を2倍、3倍…と計算していき、10000を越えたらwhileループを打ち切る。

リスト　解答

```
/* baisuu.c */
#include <stdio.h>

main()
{
    int dt,n;

    printf("数値(10000以下)=");
    scanf("%d",&dt);                ……数値の入力
    n=1;
    while(dt*n <= 10000){           ……数値dtのn倍数が10000以内の間ループする
        n=n+1;
    }
    printf("10000を越えない最大の倍数は %d¥n",dt*(n-1));
}
```

実行結果

```
C>baisuu
数値(10000以下)=30
10000を越えない最大の倍数は 9990

C>baisuu
数値(10000以下)=70
10000を越えない最大の倍数は 9940

C>baisuu
数値(10000以下)=329
10000を越えない最大の倍数は 9870
```

例題４：九九の表を表示する

1～9は周知なので、ここでは10から20までの掛け算の結果を表示する。
このプログラムは多重ループ（ループの中にまたループがある）の例である。

リスト　解答

```
#include <stdio.h>

main()
{
    int i,j;
```

```
    printf("     10  11  12  13  14  15  16  17  18  19  20¥n");
    for(i=10; i<=20; i++){
        printf("¥n%2d ",i);
        for(j=10; j<=20; j++){
            printf("%4d",i*j);
        }
    }
}
```

実行結果

```
     10  11  12  13  14  15  16  17  18  19  20

10  100 110 120 130 140 150 160 170 180 190 200
11  110 121 132 143 154 165 176 187 198 209 220
12  120 132 144 156 168 180 192 204 216 228 240
13  130 143 156 169 182 195 208 221 234 247 260
14  140 154 168 182 196 210 224 238 252 266 280
15  150 165 180 195 210 225 240 255 270 285 300
16  160 176 192 208 224 240 256 272 288 304 320
17  170 187 204 221 238 255 272 289 306 323 340
18  180 198 216 234 252 270 288 306 324 342 360
19  190 209 228 247 266 285 304 323 342 361 380
20  200 220 240 260 280 300 320 340 360 380 400
```

この章のまとめ

- 同じ処理を繰り返すときは**for**文、**while**文を使う
- 繰り返し回数がはじめからわかっているときは**for**文を使う
- **for**文でn回ループさせるには、

  for(i=1; i<=n; i++)

  の決まり文句を応用して使うとよい
- 繰り返し回数がわかっていないときは**while**文を使う
- **while**文では条件要素の初期設定とループ内再設定がポイントである
- 数字表示のとき表示幅を指定するときは次のようにする

```
printf("%5d¥n",a);
```
……aの値を5文字幅で表示

用例で学ぶ

- for文とwhile文の記述例を以下に示す。

記述	説明
`for(i=1; i<=10; i++)` `    sum=sum+i;`	iの値が1から10になるまでループ 単文なので{}は不要（あえて付けてもよい）

`for(i=1; i<=10; i++){`
`    sum=sum+i;`
`    mul=mul*i;`
`}`
	複数の文なので{}が必要

`for(i=0; i<10; i++)` `    sum=sum+i;`	これは0〜9までのループになる "<="でなく"<"になっているのに注意

`for(i=10; i<=20; i=i+2){`
`    ...`
`}`
	刻み幅2のfor文

`for(i=10; i<=2000; i=i*2){`
`    ...`
`}`
	刻み幅2倍のfor文

`for(i=20; i>=1; i--){`
`    ...`
`}`
	20から1までのカウントダウンfor文
	"<="でなく">="であることに注意
	i++ではなくi--になる

`while(a[i]!=0)` `    i++;`	制御される文が1つのときは{}は不要 （あえて付けてもよい）

`while(sum<=1000){`
`    sum=sum+dt;`
`    ++dt;`
`}`
	制御される文が複数のときは{}が必要

`while(1){`
`    ...`
`}`
	無限ループの記述

Chapter 10

よく使う処理は関数にしよう（関数）

10.1 関数というもの

我々は日常生活で、同じ処理パターンを複数回行なうことがあります。

たとえば、ジュース自動販売機を前にして「お金」を入れ、「ボタン選択」という情報を渡してやると、コーラやウーロン茶という「飲物」が戻ってきます。

話を明確にするため、このときボタンに1〜10の番号が付いていたとします。するとこの行為は次のように表現できます。

飲物　←　ジュース自動販売機（お金、番号）

もっと数字寄りの例をあげてみましょう。電卓をもった少しばかり病的な男性がいます。彼に数字を教えると、彼は「数字×数字×3.14159」の計算（つまり円の面積）をして、答えを教えてくれるものとします。このように男は半径を与えられると、円の面積だけを忠実に計算します。この行為は次のように表現できます。

円の面積　←　円面積計算男（半径）

このように、

何かを入れると、ある厳密なルールに従って、何かが戻ってくる

という一定の関係にあるものを、Cでは「関数」と呼んでいます。なお、他のプログラミング言語では、値を返すものを「関数」、同じ構造をもつが値を返さないものを「サブルーチン」と使い分けることがあります。Cは両者を統合して単に関数と呼んでいます。

ジュース自動販売機と円面積計算男の関数的関係

●ジュース自動販売機

ジュースが出力される

お金と番号を与える

取出口

●円面積計算男

円の面積が出力される

半径を手渡す（入力する）

円の面積は
314.159..

半径
10

10.2 ベキ計算を行なう関数

次の関数bekiはベキ（累乗）計算を行なうものです。

```
a=beki(2,3);
```

とするとaに8（2の3乗）が入ります。

リスト　関数でベキ計算をする

```
 1: #include <stdio.h>
 2: int beki(int a, int b);              ……関数bekiのプロトタイプ
 3:
 4: main()
 5: {
 6:     int n;
 7:
 8:     n=beki(2,3);                     ……2の3乗を計算して
 9:     printf("%d¥n",n);                ……その結果を出力
10:
11:     printf("%d¥n",beki(3,4));        ……直接書くこともできる
12: }
13:
14: int beki(int a, int b)               ……関数beki本体
15: {
16:     int i,ans;
17:
18:     ans=1;                           ┐
19:     for(i=1; i<=b; i++){             │
20:         ans=ans*a;                   ├aのb乗を計算
21:     }                                ┘
22:     return ans;                      ……結果を戻す
23: }
```

実行結果

```
8      ……2の3乗
81     ……3の4乗
```

関数の構成

関数を記述するときは、次のことを決めなくてはいけません。

関数の名前は何にするか
関数に渡すデータの型と名前は何か
関数から戻るデータ（戻り値）の型は何か
関数本体の処理方法の記述
結果を戻すための記述

ここで「関数に渡すデータ」のことを「引数」と呼んでいます。この関数bekiは次のようになっています。

```
int beki(int a, int b)
{
    int i,ans;

    ans=1;
    for(i=1; i<=b; i++){
       ans=ans*a;
    }
   return ans;
}
```

戻り値はint型（=変数ansの型）である
関数名はbekiである
最初の引数はint型（その名前を仮にaとする）である
2番目の引数はint型（その名前を仮にbとする）である
関数本体はここから
この関数の中で使用する変数の宣言
渡されたデータa、bを使って戻り値ansを計算
ansの値を戻す
関数本体はここまで

関数からの戻り値

戻り値を呼びだし側に返すときはreturn文を用います。

```
return ans;
```

で変数ansの内容が戻り値として、関数呼びだし側に戻ります。戻り値は変数である必要はなく、

```
return 10;
```

のような記述もできます。なおreturn文を書くとき、関数と表現形式を合わせて、

```
return(10);
```

のようにかっこを付ける人もいます。どちらでもかまいませんが、returnは関数ではないので、本来このかっこは不要です。

戻り値のないreturn

void型関数（127ページで説明）の場合は、戻り値は必要ありません。このときreturnを途中脱出に使うことができます。

```
void myfunc(int dt)
{
    if(dt<=0) return;  /*dtが0以下なら処理を打ち切る*/
    ...
}
```

複数のreturn

returnは関数の中に何回でてきてもかまいません。最初に出会ったreturn文で、その関数を終了します。

```
int min(int a, int b) /* 小さい数値を返す */
{
    if(a<b)                  ……a<bなら
        return a;            ……aの値を戻してこの関数を終了
    else                     ……そうでないなら
        return b ;           ……bの値を戻してこの関数を終了
}
```

beki関数の呼びだし

メインルーチン

文

文

n=beki(2,3);

文

文

beki関数

2,3

文

文

return ans;

8

関数の仕様（プロトタイプ）を前もって書く

関数を作ったときはその仕様を先頭部分（main関数より前）に書くようにします。それが2行目の、

```
int beki(int a, int b);
```

です。これは見てわかる通り関数の先頭行、すなわち14行目の、

```
int beki(int a, int b)
```

をそっくりもってきて「最後に;を付加した」ものです。これを関数プロトタイプと呼んでいます。関数プロトタイプを書いておくと、

```
a=beki(10);    ……引数が1つしかない
```

といった誤った利用をしたときコンパイラが警告してくれます。この例のプロトタイプは、

――関数bekiが使われたとき「その戻り値がint型であること、引数が2個であること、最初の引数がint型であること、2番目の引数がint型であること」をチェックする。これに違反したときは警告する

という役割をもっています。

関数プロトタイプを省略する書き方

Cコンパイラはプログラムを、先頭から1行ずつ処理していきます。そのとき、

関数を呼びだす時点で、その関数の仕様が判明していなければならない

ことになっています。このプログラムは、
——mainより後方に関数bekiがある
という位置関係になっています。そのためこのままでは、main関数の中でbeki関数を呼びだそうとするとき、その仕様が判明しません。そこで、
——じゃあ、その関数bekiの仕様だけはmainの前に書いておこう
としています。それがプロトタイプです。こう述べると、
——じゃあmainを最後に書けば、プロトタイプはいらないじゃないか
という考えがでてきます。Cでは関数を書く順序は規定されていませんので、これはその通りです。次のリストは同じプログラムをプロトタイプ不要スタイルで書いたものです。

リスト 関数でベキ計算をする(プロトタイプ不要スタイル)

```
 1: #include <stdio.h>
 2:                                    ……プロトタイプが不要。なぜなら
 3: int beki(int a, int b)             ……関数bekiを先に書いたから
 4: {
 5:     int i,ans;
 6: 
 7:     ans=1;
 8:     for(i=1; i<=b; i++){
 9:         ans=ans*a;
10:     }
11:     return ans;
12: }
13: 
14: main()                             ……mainを最後に書く
15: {
16:     int n;
17: 
18:     n=beki(2,3);                   ……この時点でbekiの仕様は判明している
19:     printf("%d¥n",n);
20: 
21:     printf("%d¥n",beki(3,4));
22: }
23: 
```

どちらのスタイルがよいのか

ここに示したようにプロトタイプに関しては2つの書き方があります。では、どちらのほうがよいのかということになりますが、

正当な書き方はやはりプロトタイプを書くスタイル

です。なぜなら、

——プログラム先頭部を見るだけで、そのプログラムの中で使われている関数に関する情報一覧がわかる

からです。そのほかのメリットもいくつかあります。

これに対してmain最終スタイルで書くと、プログラムが大きくなり自作関数が多くなったとき、関数の相互関係が入り組んできて、どの関数を前にださなければならないかがわかりにくくなってきます。また複数の関数が、互いに相手を呼び合っている場合は記述不能になります。たとえば、

```
int func1(int dt)
{
    ……
    n=func2(10);
}

int func2(int dt)
{
    ……
    n=func1(10);
}
```

では、どちらの関数を前にだしても不都合になります。このプログラムは、

```
int func1(int dt);
int func2(int dt);

int func1(int dt)
{
    ……
    n=func2(10);
}

int func2(int dt)
{
```

```
    ……
    n=func1(10);
}
```

のように、プロトタイプを先頭に置いたときに、はじめて有効なプログラムになります。結局、あるサイズ以上のプログラムになるとプロトタイプをしっかり書いておいたほうが、さまざまな利点があるわけです。

ただ、初心者にとってプロトタイプを正しく書くことは、いくらか負担になります。またプログラムが小さいときは、全体の見通しもよく、「プロトタイプが書いてあったから関数の仕様がすぐわかって助かった」ということもあまりありません。

そこで、

――初心者が小さなプログラムを書くときは「main最後スタイル」でもかまわない

というのも現実的対応です。

本書でも小さなプログラムが主体ですので、以後、

main最後スタイル

で書くことがあります。

実数を扱う関数

引数や戻り値に実数を使う関数を作るのも簡単です。データ型をfloatにするだけです。次の関数en_mensekiは引数も戻り値もfloat型を使用したものです。半径を与えるとその面積を返します。

```
float en_menseki(float r)
{
    return r*r*3.14159;
}
```

10.3 戻り値のない関数

Cではもし戻り値が不要なら、戻り値のない関数を書くこともできます。次のプログラムは数字を出力するものです。変数dtの値を10進数で出力する場合、毎回、

```
printf("%d¥n",dt);
```

と書かなければなりません。これは多少わずらわしいことなので、

```
putd(dt);
```

で済む関数を作ることにしましょう。この関数は与えられた引数値を出力するだけで、戻り値はありません。

リスト　10進数を表示する関数putd

```
 1: #include <stdio.h>
 2:
 3: void putd(int a)              ……引数を10進数出力する関数。voidに注目
 4: {
 5:     printf("%d¥n",a);         ……引数aの値を出力する
 6: }
 7:
 8: main()
 9: {
10:     int dt=5000;
11:
12:     putd(1000);               ……1000を出力
13:     putd(dt);                 ……dtの値を出力
14: }
```

実行結果

```
1000
5000
```

void宣言

上のプログラムの3〜6行目が戻り値のない関数の記述例です。この関数putdは与えられたデータの値を10進数出力するだけです。何の戻り値もありません。このように戻り値のない関数の場合は、
――戻り値がない
という意味の宣言をしておきます。それが、

```
void
```

という記述です。この他、引数がないときにもvoidという表現を使います。関数の引数と戻り値のパターンを示すと、次の図のようになります。

関数の引数と戻り値のありなし例

引数あり → / 戻り値あり ←

```
int smp(int a)
{
    ...
}
```

引数あり → / 戻り値なし

```
void smp(int a)
{
    ...
}
```

引数なし / 戻り値あり ←

```
int smp(void)
{
    ...
}
```

引数なし / 戻り値なし

```
void smp(void)
{
    ...
}
```

10.4 例題プログラム

例題

円の面積を計算する。4つのパターンの関数を作成する

リスト 解答

```
#include <stdio.h>

void opening(void)                          /* 戻り値なし 引数なし */
{
    printf("円の面積を計算します¥n");
}

float getdata(void)                         /* 戻り値あり 引数なし */
{
    float wk;
    printf("半径を入力してください: ");
    scanf("%f",&wk);
    return wk;
}

float en_menseki(float r)                   /* 戻り値あり 引数あり */
{
    return r*r*3.14159;
}

void dispdata(float dt)                     /* 戻り値なし 引数あり */
{
    printf("面積 = %f¥n",dt);
}

main()
{
    float hankei,menseki;

    opening();
    hankei=getdata();
    menseki=en_menseki(hankei);
    dispdata(menseki);
}
```

実行結果

```
円の面積を計算します
半径を入力してください：3.5
面積 = 38.484478
```

この章のまとめ

- 何度も再利用する処理は関数にするとよい
- 関数にはデータを渡すことができる。この引き渡しのためのデータを引数と呼ぶ
- 関数から戻り値を得ることができる
- 戻り値と引数はそのデータの型を指定する
- 必要なら戻り値のない関数や、引数のない関数を作ることができる
- 戻り値や引数がないときは**void**と記述する
- 関数の一般型の例は次の通り

関数の種類	一般型
戻り値あり引数あり	int　smp(int a)
戻り値あり引数なし	int　smp(void)
戻り値なし引数あり	void smp(int a)
戻り値なし引数なし	void smp(void)

- 関数を記述したときはそのプロトタイプを**main**より前に書かなくてはならない

```
#include <stdio.h>
int beki(int a, int b);          ……プロトタイプを書く

main()
{
    ...
}

int beki(int a, int b)
{
    ...
}
```

- ただし**main**を最後に書くときはこのプロトタイプは不要である

```
#include <stdio.h>

int beki(int a, int b)
{
    ...
}

main()                              ……mainを最後に書く
{
    ...
}
```

- 小さなプログラムなら「**main**最後スタイル」でもいいが、通常はプロトタイプを書くほうがよい

Chapter 11
できあいの関数を使おう（標準関数）

11.1 ステートメントと関数

プログラミング言語には動作を指定する方法としてステートメント（文）が用意されています。その代表例が制御文で、Cでは、

```
if else
for
while
switch
```

などがさかんに用いられます。また、なんらかの値を返す「関数」と呼ばれる機能も用意されています。C言語はステートメントと関数を使ってさまざまなプログラムを作っていきます。

一般にFORTRANやBASICなどの言語では、データの入出力を行なう機能（たとえばPRINT文）も、ステートメントの一部として用意されています。しかしCは少し異なります。Cではif、for、whileなどの制御文で流れを制御しますが、それ以外の特定の仕事を行なわせる場合は、一貫して関数を用います。たとえば文字列をコピーするstrcpyや、入出力を行なうprintf、scanfなども、すべて関数です。

関数には2種類あります。それは、

(1) 自分で作る関数
(2) はじめから用意されている標準関数

です。標準関数はCコンパイラを購入すると、その中にはじめから用意されています。strcpyやprintfやscanfも含まれます。標準関数を使うということは、Cコンパイラ開発者が用意してくれた、便利な機能を活用するということです。

11.2 文字列の処理

Cでは文字列の処理用に標準関数が用意されています。ここでは数字と並んで、代表的なデータである文字列を、標準関数を使ってどう処理するかについて説明します。文字列処理には、次の4つの主機能があります。

文字列処理の機能

処理1：文字列の代入（strcpy）

代入 "abcdef"

処理2：文字列の長さを求める（strlen）

長さ=6
"abcdef"

処理3：文字列を比較する（strcmp）

"abcdef" VS "uvwxyz"

処理4：文字列を連結する（strcat）

"abcdef" + "uvwxyz"

次のプログラムは文字列に対する、

代入　長さ取得　比較　連結

という働きを確認するものです。

リスト 文字列の代入、長さ取得、比較、連結を行なう

```
 1: #include <stdio.h>
 2: #include <string.h>                ……文字処理関数用プロトタイプ
 3: main()
 4: {
 5:     char s1[80],s2[80];            ……文字列用変数
 6:     int ln;
 7:
 8:     strcpy(s1,"abcdef");           ……文字列の代入(コピー)
 9:     strcpy(s2,"uvwxyz");
10:     printf("s1=%s s2=%s¥n",s1,s2);
11:
12:     ln=strlen(s1);                 ……文字列の長さを求める
13:     printf("len=%d¥n",ln);
14:
15:     if(strcmp(s1,s2) > 0)          ┐
16:         printf("左辺が大¥n");       │
17:     if(strcmp(s1,s2) < 0)          │
18:         printf("右辺が大¥n");       │
19:     if(strcmp(s1,s2) == 0)         ├文字列を比較する
20:         printf("等しい¥n");         │
21:     if(strcmp(s1,s2) != 0)         │
22:         printf("等しくない¥n");     ┘
23:
24:     strcat(s1,s2);                 ……文字列を連結する
25:     printf("s1+s2=%s¥n",s1);
26: }
```

実行結果

```
s1=abcdef s2=uvwxyz
len=6
右辺が大
等しくない
s1+s2=abcdefuvwxyz
```

文字処理関数用のプロトタイプ

Cの関数には、それぞれ必要なプロトタイプ(を記述したファイル)が用意されています。これを用いることにより、その関数を不当に使用したときに警告してくれます。strcpy、

strlenといった文字列処理関数のプロトタイプは、一括してstring.hというファイルの中に記述されていますから、これを使うときは2行目のように、

```
#include <string.h>
```

と書きます。こうすれば文字処理関数を何種類使っても、何度使ってもかまいません。どの関数がどのプロトタイプを必要とするかは、それぞれのC言語処理系のマニュアルに載っています。

文字列の代入

書式　strcpy(代入される文字列変数, 代入する文字列);

文字列の代入にはstrcpy関数を使います。この関数の機能は「代入」ですが、Cでは「文字列をコピーする」と表現することもあります。次に使用例を示します。

```
strcpy(a,"ABCDE");    ……文字列変数aは"ABCDE"になる
strcpy(a,b);          ……文字列変数aに文字列変数bの文字列を代入する
```

文字列の長さを求める

書式　strlen(文字列);

文字列の長さを求めるときはstrlen関数を使います。この関数は長さを戻り値として返します。次に使用例を示します。

```
n=strlen("ABCDE");    ……文字列"ABCDE"の長さを求める。nは5になる
n=strlen(a);          ……文字列変数aで示す文字列の長さを求める
```

文字列の比較

書式　strcmp(文字列1, 文字列2);

文字列の比較にはstrcmp関数を使います。この関数では文字列の比較は辞書順（文字コード順）で行なわれます。英文字の場合は、

――辞書に先にでてくる文字列のほうが小さい

となります。なお大文字は小文字より小さいと判断されます（参考：'A'の文字コードは65、'a'の文字コードは97）。strcmp関数は比較結果を次の数値で返します。

strcmp(a, b);の結果

戻り値	内容
正	a > b
0	a == b
負	a < b

strcmp関数は通常はif文と一緒に使用します。そのとき比較は次のように行ないます。

strcmp関数による比較

記述	意味
if(strcmp(a,b) == 0)	文字列aと文字列bが　等しいなら
if(strcmp(a,b) != 0)	文字列aと文字列bが　等しくないなら
if(strcmp(a,b) > 0)	文字列aが文字列bより 大きいなら
if(strcmp(a,b) >= 0)	文字列aが文字列bより 大きいか等しいなら
if(strcmp(a,b) < 0)	文字列aが文字列bより 小さいなら
if(strcmp(a,b) <= 0)	文字列aが文字列bより 小さいか等しいなら

次にstrcmpの使用例をあげます。

```
if(strcmp(a,"ABCDE") == 0) ……文字列変数aと文字列"ABCDE"を比較する
if(strcmp(a,b) == 0)       ……文字列変数aと文字列変数bを比較する
```

文字列の連結

書式　**strcat(文字列変数, 連結する文字列);**

文字列の連結にはstrcat関数を使います。連結処理の結果、第1引数で示す文字列変数の内容が、連結後の文字列になります。

次に使用例を示します。文字列変数aの最初の値は"XYZ"とします。

```
strcat(a,"ABCDE");   ……文字列変数aは"XYZABCDE"になる
strcat(a,b);         ……文字列変数aに文字列変数bの文字列を連結する
```

11.3 数学関数を使う

Cには数学処理用に数学関数と呼ばれる関数群が用意されています。次のプログラムは10度から80度までのサイン値一覧表を出力するものです。ここではsinという関数を利用しています。

リスト　10度から80度までのサイン値一覧表を出力する

```
 1: #include <stdio.h>
 2: #include <math.h>                ……サイン関数用プロトタイプ
 3: main()
 4: {
 5:     int i;
 6:     float d;
 7:
 8:     for(i=10; i<=80; i=i+10){
 9:         d=sin(i*3.14159/180.0);  ……sin関数
10:         printf("%d  %f¥n",i,d);
11:     }
12: }
```

実行結果

```
10  0.173648
20  0.342020
30  0.500000
40  0.642787
50  0.766044
60  0.866025
70  0.939692
80  0.984808
```

sin関数

Cのsin関数は9行目のように記述します。用法は、

結果=sin(角度);

です。結果は実数で戻りますのでfloat型の変数で受けます。また精度がほしいときは

double型の変数を使用します。double型については「Chapter13　その他のCの機能を見てみよう（C言語の全体像）」（→166ページ）を参照してください。

ラジアン変換

sin関数の角度は一般の度数ではなくラジアン値で与えます。これはプログラミング言語（BASICやFORTRANなど）では皆同じです。サイン30度の値を求めたいときは、

```
d=sin(30.0);
```

ではなく、

```
d=sin(30.0*3.14159/180.0);
```

とします。ラジアンは360度を、

$2\pi = 6.28318$ラジアン

とする平面角単位です。すなわち1ラジアンは約57.3度となります。

sin30度の求め方

求めたい角度 ► 30°

ラジアン変換する ► 30.0*3.14159/180.0

sin関数に当てはめる ► d=sin(30.0*3.14159/180.0);

sin関数用のプロトタイプ

sin関数のプロトタイプはmath.hファイルに入っています。これを使用するときは2行目の、

```
#include <math.h>
```

のように書きます。sin、cos、tanといった数学用の関数はすべてmath.hというファイルに入っている関数プロトタイプを使います。

代表的な数学関数の利用例

代表的な数学関数の利用例を示します。それぞれの関数の用法は次の通りです。

```
サイン値        = sin(ラジアン角度);
コサイン値      = cos(ラジアン角度);
タンジェント値  = tan(ラジアン角度);
ベキ乗値        = pow(x,y);          ……xのy乗計算
常用対数値      = log10(x);          ……xの常用対数計算
平方根          = sqrt(x);           ……xの平方根計算
```

リスト　サイン値、コサイン値、タンジェント値、ベキ乗値、常用対数値、平方根を計算する

```
 1: #include <stdio.h>
 2: #include <math.h>                 ……プロトタイプを一度だけ書く
 3: main()
 4: {
 5:      printf("sin 30°=%f¥n",sin(30.0*3.14159/180.0));
 6:      printf("cos 30°=%f¥n",cos(30.0*3.14159/180.0));
 7:      printf("tan 45°=%f¥n",tan(45.0*3.14159/180.0));
 8:      printf("2の3乗 =%f¥n",pow(2.0, 3.0));
 9:      printf("1000の常用対数=%f¥n",log10(1000.0));
10:      printf("10の平方根=%f¥n",sqrt(10.0));
11: }
```

実行結果

```
sin 30°=0.500000
cos 30°=0.866026
tan 45°=0.999999
2の3乗 =8.000000
1000の常用対数=3.000000
10の平方根=3.162278
```

11.4 MS-DOSコマンドを実行する

MS-DOSには、

DIR
COPY

といった便利なコマンドがあります。このようなコマンドがプログラムの中から実行できれば便利です。Cにはそれを行なう便利な関数systemが用意されています。次のプログラムはMS-DOSのコマンドラインから、

```
TIME
DATE
VER
```

と入力したのと同じ働きをします。

リスト　MS-DOSコマンドをCプログラムから実行する

```
1: /* syststst.c */
2: #include <stdio.h>
3: #include <stdlib.h>          ……system関数のプロトタイプが入っている
4: main()
5: {
6:     system("time");
7:     system("date");
8:     system("ver");
9: }
```

実行結果（Windows95のDOSウィンドウで実行）

```
現在の時刻は  16:48:55.74
時刻を入力してください： ↵
現在の日付は 1998-05-14 (木)
日付を入力してください(年-月-日)： ↵

Windows 95. [Version 4.00.1111]
```

system関数

system関数はプログラムの中からOSのコマンドを実行させたいときに使います。書式は、

書式

```
system("実行させたいコマンド");
```

となります。たとえばディレクトリ情報を知りたいとき、わざわざ自分で同じ機能のものを作る必要はありません。次の1行を書けばいいのです。

```
system("dir");
```

system関数のプロトタイプ

system関数のプロトタイプはstdlib.hに入っています。したがって、この関数を使用するときは、

```
#include <stdlib.h>
```

と書いてやります。

COMMAND.COMが必要

system関数はいきなりDIRなどのMS-DOSコマンドを実行するわけではありません。DIRコマンドは実際にはMS-DOSのCOMMAND.COMというプログラムの中に入っている機能です。system関数はDIRなどのMS-DOSコマンドを実行するのに、一度COMMAND.COMプログラムを読み込みます。そのため、system関数を実行するときCOMMAND.COMプログラムが適切な場所（現在のディレクトリなど）にあることが必要となります。

systst.cに見るsystem関数の働き

MS-DOS

syststプログラム

C>

プロンプト状態

C>systst

プログラム起動

system("time");

system関数による
MS-DOSコマンド実行

TIMEコマンド実行

プログラムに戻る

system("date");

DATEコマンド実行

プログラムに戻る

system("ver");

VERコマンド実行

プログラムに戻る

プログラム終了

MS-DOSに戻る

C>

11.5 その他の標準関数

Cでは制御文を使ったり、演算子を使うとき以外の処理は皆、標準関数(と自作関数)で実現されます。これまで特に説明もなく使用した、

gets　puts　scanf　printf　getchar　putchar

といった関数もすべて標準関数です。次の章で説明する、

```
fopen   fclose   fgets   fputs
```

といったファイル処理用の関数も標準関数です。また1文字単位で処理する次のような便利な関数も揃っています。

文字処理用関数

関数名	機能
isalnum(c)	文字cが英数字(A～Z、a～z、0～9)なら真
isalpha(c)	文字cが英文字(A～Z、a～z)なら真
isdigit(c)	文字cが数字(0～9)なら真
islower(c)	文字cが小文字(a～z)なら真
isupper(c)	文字cが大文字(A～Z)なら真
isxdigit(c)	文字cが16進表示文字(0～9、A～F、a～f)なら真
tolower(c)	文字cが大文字なら小文字に変換する
toupper(c)	文字cが小文字なら大文字に変換する

どのような標準関数があり、そのプロトタイプはどのファイルに入っているか、関数の使い方はどうするか、などはCコンパイラのマニュアルに載っています。

この章のまとめ

- Cでは標準的な処理を行なうために標準関数が用意されている
- 標準関数を使うときはそれぞれ用意されたプロトタイプを用いる
- たとえば、**sin**、**cos**、**tan**などの数学用関数には**math.h**というプロトタイプが用意されている
- プロトタイプを使うときは、格納されているファイルを指定して次のように書く

```
#include <math.h>
```

- どのような標準関数があるか、どのプロトタイプが必要かはマニュアルに記載されている
- 関数の名前は「**str**～」や「**is**～」のように、性格により一貫しているものが多い
- Cプログラムの中からMS-DOSコマンドを実行するときは**system**関数を使う
- **system**関数を使うときは**COMMAND.COM**が必要である(MS-DOSの場合)

Chapter 12
ファイル処理も覚えよう（ファイル処理）

12.1 データ処理の流れ

コンピュータで何らかのデータを処理する場合、その処理の流れには次の4パターンが考えられます。いずれもデータをどこから入手し、どこへだすかという問題です。

データ処理の4つの流れ

1
キーボードから入力
データ処理
ディスプレイに出力

2
キーボードから入力
データ処理
ファイルに出力

3
ファイルから入力
データ処理
ディスプレイに出力

4
ファイルから入力
データ処理
ファイルに出力

これまでの説明は①のキーボードからディスプレイへという流れだけでしたが、これでは不十分です。なぜならキーボードやディスプレイではデータの保存ができず、一発勝負のデータ入出力になってしまうからです。本当にコンピュータでデータ処理をしようとしたら、やはり、

・ファイルからの読み込み
・ファイルへの書き込み

の方法を知らなければなりません。つまり「データを保存する」ということを考える必要があります。ここではそれを説明することにします。

12.2 ファイルから読み込む

次のプログラムはファイルからデータを1行ずつ読み込み、画面に出力するものです。これはMS-DOSのTYPEコマンドと似た働きをします。プログラムの名前をtyp.cとします。

リスト　MS-DOSのTYPEコマンドと同じ働きをするプログラム

```
 1: /* typ.c */
 2: #include <stdio.h>
 3: #include <stdlib.h>  /* exit */
 4:
 5: main()
 6: {
 7:     FILE *fin;
 8:     char buf[256];
 9:     char filename1[20];
10:
11:     printf("入力ファイル名=");
12:     gets(filename1);
13:
14:     fin=fopen(filename1,"r");
15:     if(fin == NULL){
16:         printf("%sがオープンできません.¥n",filename1);
17:         exit(1);
18:     }
19:
20:     while(fgets(buf,256,fin) != NULL){
21:         printf("%s",buf);
22:     }
23:
24:     fclose(fin);
25: }
```

実行結果

```
C>type text1           ……text1というファイルがある
sample file
abcdefghij             MS-DOSのTYPEコマンドで表示されたtext1の内容
1234567890

C>typ                  ……typ実行
ファイル名=abc          ……存在しないファイルを指定した
abcがオープンできません.

C>typ
ファイル名=text1        ……ファイルtext1を表示しろ
sample file
abcdefghij             ファイルtext1の内容が表示される
1234567890
```

読み込み用ファイルをオープンする

ファイルとの間で読み書きをするためには、まずファイルをオープンしなければなりません。ここでファイルのオープンとは、

ファイルオープン → ファイル読み書きのための内部的な準備

と考えることができます。この処理は14行目のように記述します。ファイルオープン処理機能は、次のようになります。

```
fin=fopen(filename1,"r");
```

- filename1で示すファイルを
- 読み込み(r:read)用に
- オープンし
- ファイルポインタfinで管理しろ

ファイルオープンのときは、読み込みなのか書き込みなのかを指定する必要があります。ここでは"r"で読み込みを指定しています。これは"read"を意味しています。

Cではオープンしたファイルは、プログラマが指定したファイルポインタと呼ばれる変数で管理します。ファイルポインタはFILEという特別のデータ型で宣言します。ファイルポインタの名前(この例ではfin)は自由に付けることができます。この後はファイルポインタを使って、

──fin(で示すファイル)からデータを読み込め
という処理を行ないます。

エラー処理

Cではファイルオープンがうまくいかないときは、自分で適切なエラー処理をする必要があります。Cには「エラーのとき自動的に何かをしてくれる」というサービスはありません。それはすべてプログラマが行ないます。そのためCプログラムのファイルオープンの一般形は14〜18行目のようになります。その意味は次の通りです。

```
fin=fopen(filename1,"r");          ── filename1を読み込み用にファイルオープンし
if(fin == NULL){                   ── もし その戻り値がNULL(=エラー)なら
    printf("%sがオープンできません.¥n",filename1); ── このエラーメッセージを表示し
    exit(1);                       ── プログラムを終了しろ
}
```

fopen関数はファイルオープン処理に失敗するとNULLで表現される値(記号定数といいます→176ページ「13.9 プリプロセッサ命令」を参照)を合図として返します。このNULLの値はstdio.hファイルの中で定義されています。プログラマはその値が何であるかは気にすることなく、
──NULLが返ってきたらエラーが起きたという合図なのでそれなりの処理をする
と考えればいいのです。エラーが起きたら必要なエラーメッセージを表示した後、

```
exit(1);        ……プログラムの強制終了
```

として、このプログラムを強制的に終了させます。

特別の希望がないならファイルオープン処理はこの例の通りに決まり文句で書くことができます。初心者はここに示したスタイルをそっくり使えばいいのです。finとfilename1という変数名は、都合のよい名前にしてかまいません。

なおexit関数のプロトタイプはstdlib.hファイルに入っているので、2行目でそれをインクルードしています。

NOTE

ファイルオープン処理の慣例記法

ファイルのオープンとそのエラー処理は、慣れてくると次のようにコンパクトに書くのが普通である。

```
if((fin=fopen(filename1,"r")) == NULL){
    printf("%sがオープンできません.¥n",filename1);
    exit(1);
}
```

データの読み込み

ファイルオープンが正しく行なわれると、そのファイルからデータを読み込むことができます。その方法にも決まり文句があります。それが20～22行目の記述です。ファイルから文字列を1行だけ読み込むには、

```
fgets(buf,256,fin);
```

とします。その意味は次の通りです。

```
fgets(buf,256,fin);
```

- fin：finで示すファイルから
- 256：最大256文字を
- buf：変数bufに
- fgets：1行読み込め

ここでは最大文字数として256を指定するため、変数bufも、

```
char buf[256];
```

として、その文字列を格納可能にしています。

読み込むファイルはファイルポインタfinで指定します。このfinは先のファイルオープン処理のときに、変数filename1と結び付けられています。「finから読み込め」というのは「finで管理されているfilename1で示すファイルから読み込め」という意味になります。

読み込み終了の確認

ファイルからデータを読み込んでいくと、やがてデータはなくなります。そのとき、
——データ読み込みが終了した
ということを知らなくてはいけません。そのためにfgets関数は、
——ファイル終了ならNULLを返す
という仕様になっています。そこで「ファイル終了まで何かを行なう」という記述は次のようにします。


```
while( fgets(buf,256,fin) != NULL  ){
    printf("%s",buf);
}
```

これで「すべてのデータを読み込め」という処理になります。
同じNULLでも、fopenの戻り値がNULLのときは「ファイルオープン失敗」、fgetsの戻り値がNULLのときは「ファイル終了」という意味なので注意してください。

ファイルをクローズする

ファイルとのデータ入出力が終了したら、そのファイルはクローズしなければなりません。ここでファイルのクローズとは、

ファイルクローズ → ファイル読み書き終了時の内部的な後始末

と考えることができます。Cでファイルをクローズするときはfclose関数を用います。24行目の、

```
fclose(fin);
```

で、finが管理しているファイル（＝fopenでオープンしたファイル）をクローズします。

12.3 ファイルに書き込む

次のプログラムはtyp.cを改造したものです。ファイルから1行ずつ読み込んだデータを別のファイルに書き込みます。これはMS-DOSのCOPYコマンドに似た働きをしま

す。プログラムの名前をcpy.cとします。

リスト MS-DOSのCOPYコマンドと似た働きをするプログラム

```
 1: /* cpy.c */
 2: #include <stdio.h>
 3: #include <stdlib.h>  /* exit */
 4:
 5: main()
 6: {
 7:     FILE *fin,*fout;
 8:     char buf[256];
 9:     char filename1[20],filename2[20];
10:
11:     printf("入力ファイル名=");
12:     gets(filename1);
13:     printf("出力ファイル名=");
14:     gets(filename2);
15:
16:     fin=fopen(filename1,"r");
17:     if(fin == NULL){
18:         printf("%sがオープンできません.¥n",filename1);
19:         exit(1);
20:     }
21:     fout=fopen(filename2,"w");
22:     if(fout == NULL){
23:         printf("%sがオープンできません.¥n",filename2);
24:         exit(1);
25:     }
26:
27:     while(fgets(buf,256,fin) != NULL){
28:         fputs(buf,fout);
29:     }
30:
31:     fclose(fin);
32:     fclose(fout);
33: }
```

実行結果

```
C>type text1    ……text1ファイルの内容
sample file
abcdefghij
1234567890
```

```
C>cpy
入力ファイル名=text1        cpyプログラム実行
出力ファイル名=text2        text1をtext2にコピーする

C>type text2              ……コピーされたtext2ファイル
sample file
abcdefghij                確かにコピーされている
1234567890
```

書き込み用のファイルオープン

書き込み用のファイルオープン方法も、読み込み用オープンと同じです。読み込み指定の'r'という文字が、書き込み指定では"write"を意味する'w'となるところが異なります。また読み込み専用のファイルポインタ（ここではfout）を指定します。

```
fout=fopen(filename2,"w");
```

データの書き込み

書き込み用ファイルオープンが正しく行なわれると、そのファイルにデータを書き込むことができます。1行書き込みには28行目のようにfputs関数を使います。

```
fputs(buf,fout);
```

foutで示すファイルに
変数bufで示す文字列を
1行書き込め

その他のファイル読み書き関数

ファイルからの読み込みにはfgets関数がもっとも無難ですが、ファイルへの書き込みの場合には、fputsの他にfprintfもよく使われます。このプログラムで使ったfputsとまったく同じ動作をfprintfで記述すると、

```
fputs(buf,fout);
fprintf(fout,"%s",fout);
```

となります。これはおなじみのprintfに「fout」という出力先ファイル指示が付いただけです。その他の使用方法はprintfとまったく同じです。たとえば画面出力の、

```
printf("data=%d¥n",dt);
```

を、foutで示すファイルに書き込むときは、

```
fprintf(fout,"data=%d¥n",dt);
```

とします。代表的なファイル読み書き関数には次のものがあります。

ファイル読み書き用関数

関数名	機能
fgetc	1文字読み込み
fputc	1文字書き込み
fgets	文字列読み込み
fputs	文字列書き込み
fscanf	数字などの書式付き読み込み
fprintf	数字などの書式付き書き込み

NOTE

本書では説明しないが、ファイル処理でのファイル名指定は、コマンドラインから行なうこともできる。次に**cpy.c**をコマンドライン方式に書き直した**cpy2.c**を参考として示す。またファイルオープン処理も上級者の好む方法を示す。

```
/* cpy2.c */
#include <stdio.h>
#include <stdlib.h>  /* exit */

main(int argc, char *argv[])
{
    FILE *fin,*fout;
    char buf[256];

    if(argc!=3){                              /* パラメータの個数をチェック */
        printf("用法: cpy2 fromfile tofile¥n"); /* 用法を示す */
        exit(1);
    }

    if((fin=fopen(argv[1],"r"))== NULL){ /* 入力ファイルをオープン */
        printf("%sがオープンできません.¥n",argv[1]);
        exit(1);
```

```
    }
    if((fout=fopen(argv[2],"w")) == NULL){   /* 出力ファイルをオープン */
        printf("%sがオープンできません.¥n",argv[2]);
        exit(1);
    }

    while(fgets(buf,256,fin) != NULL){
        fputs(buf,fout);
    }

    fclose(fin);
    fclose(fout);
}
```

実行結果

```
C>cpy2
用法: cpy2 fromfile tofile          ……エラーメッセージ表示

C>cpy2 text1 text2                  ……text1をtext2にコピー

C>type text2                        ……コピーされたtext2を見る
sample file
abcdefghij
1234567890
```

12.4 例題プログラム

例題 1

ファイルの行数をカウントする。

リスト 解答

```
/* linect.c */
#include <stdio.h>
#include <stdlib.h>  /* exit */

main()
```

```
{
    FILE *fin;
    char buf[256];
    char filename1[20];
    int ct;                    /* 行数カウント用 */

    printf("入力ファイル名=");
    gets(filename1);

    fin=fopen(filename1,"r");
    if(fin == NULL){
        printf("%sがオープンできません.¥n",filename1);
        exit(1);
    }

    ct=0;
    while(fgets(buf,256,fin) != NULL){
        ++ct;                  /* 行数カウントアップ */
    }
    printf("%d lines¥n",ct); /* 行数の表示 */

    fclose(fin);
}
```

実行結果

```
入力ファイル名=linect.c    ……自分自身の行数をカウントする
28 lines
```

例題2

行番号付きでファイルを表示する。

リスト **解答**

```
/* ntype.c */
#include <stdio.h>
#include <stdlib.h>   /* exit */

main()
{
    FILE *fin;
    char buf[256];
```

```
    char filename1[20];
    int lineno;  /* 行番号カウント用 */

    printf("入力ファイル名=");
    gets(filename1);

    fin=fopen(filename1,"r");
    if(fin == NULL){
        printf("%sがオープンできません.¥n",filename1);
        exit(1);
    }

    lineno=0;
    while(fgets(buf,256,fin) != NULL){
        ++lineno;                        /* 行番号を +1 する */
        printf("%4d:%s",lineno,buf); /* 行番号とテキストを表示 */
    }

    fclose(fin);
}
```

実行結果

```
入力ファイル名=ntype.c       ……自分自身を表示させてみる
   1:/* ntype.c */
   2:#include <stdio.h>
   3:#include <stdlib.h>  /* exit */
   4:
   5:main()
   6:{
   7:    FILE *fin;
   8:    char buf[256];
         :
    （途中省略）
         :
  27:    fclose(fin);
  28:}
```

この章のまとめ

- ファイルはオープンしてから、読み書きを行ない、最後にクローズする
- ファイルのオープンには **fopen** を用いる
- ファイルに対する１行読み込みには **fgets** を､1 行書き込みには **fputs** を用いる
- ファイルのクローズには **fclose** を用いる
- ファイルの読み込みオープンには次のひな型を用いるとよい

```
if((fin=fopen(filename1,"r")) == NULL){
    printf("%sがオープンできません.¥n",filename1);
    exit(1);
}
```

- ファイルの書き込みオープンには次のひな型を用いるとよい

```
if((fout=fopen(filename2,"w")) == NULL){
    printf("%sがオープンできません.¥n",filename2);
    exit(1);
}
```

- ファイルからの１行読み込みには次のひな型を用いるとよい

```
while(fgets(buf,256,fin) != NULL){
    （何かの処理）
}
```

Chapter 13

その他のCの機能を見てみよう（C言語の全体像）

これまで、初心者が通常のプログラムを作成できるようにするための説明を行なってきました。そこでは使用頻度の少ない文法はあえて説明せず、必要不可欠な機能を選択して示してきました。そのため、たとえば変数型も、int、floatを主にしてきたわけです。

説明の最後として、本書を読まれてそのサンプルプログラムを十分理解できるようになった人が、次のステップに進むための知識を示すことにします。すなわちCの全体像を要約して示すことにします。

13.1 予約語と名前

Cでは次の用語を予約語としています。予約語はそれぞれに役目をもっています。これらの予約語を変数名や関数名に使用することはできません。またCコンパイラによっては、これ以外にも独自の予約語を定めている場合があります。

Cの予約語

```
auto        double      int         struct
break       else        long        switch
case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

変数の名前などには、この予約語を避けた名前を付けます。名前は、

下線（_）　英文字　数字

で構成します。名前の先頭の文字は下線、または英文字を使います。Cコンパイラは名前の先頭から少なくとも31文字までは識別します。

13.2 定数

数値と文字と文字列はそれぞれ次のように定数表現されます。

```
100       →  数値定数
'A'       →  文字定数
"ABCDE"   →  文字列定数
```

これは、

```
dt=100;                ……変数dtに数値100を入れる
ch='A';                ……変数chに文字Aのコードを入れる
strcpy(ss,"abcde");    ……変数ssに文字列abcdeを入れる
```

のように用いられます。

13.3 エスケープ文字列

printfの中で¥nと書くと復改動作を行ないます。この'¥'で始まる文字列のことをエスケープ文字列といいます。これは本来の文字の意味を離れた(エスケープした)特別の文字列という意味です。Cではエスケープ文字列があるので、「文字として表現できないが、文字コードとして定義されている文字」をその意味を連想させる表現で使うことができます。よく使われるエスケープ文字列をまとめると次のようになります。

エスケープ文字列

文字列	文字コード	機能
¥a	07	内蔵ベルを鳴らす(alert)
¥n	0A	復帰改行(new line)
¥t	09	水平タブ(tab)
¥¥	5C	¥自身を出力
¥'	2C	'を出力
¥"	22	"を出力

13.4 データ型の種類

これまでintとfloatを主体に説明しましたが、覚えておくといいものとして次のデータ型があります。

Cの主要なデータ型

データ型	表現	範囲
char	文字型	−128～127
int（16ビット）	整数型	−32768～32767
int（32ビット）	整数型	−2147483648～2147483647
long	倍長整数型	−2147483648～2147483647
float	単精度実数型	およそ10^{-38}～10^{38}（有効桁7桁）
double	倍精度実数型	およそ10^{-308}～10^{308}（有効桁15桁）

表中で「範囲」というのは、その変数を使って、どれだけの数値を操作することができるかです。この表現範囲はCコンパイラによって異なることがあります。特にint型はCコンパイラによって16ビットサイズのものと32ビットサイズのものがあります。たとえば従来のMS-DOS上で用いるCコンパイラは16ビットサイズであり、Windows上で使うVisual C++（に含まれるMicrosoft C/C++）の場合は32ビットサイズです。

この他にも、使用頻度は少なくなりますが次のデータ型があります（表現範囲は一例）。

Cの上記以外のデータ型

データ型	範囲
unsigned char	0～255
short	−32768～32767
unsigned short	0～65535
unsigned int	0～65535 または 0～4294967295
unsigned long	0～4294967295
long double	およそ10^{-4932}～10^{4932}（有効桁19桁）

各種のデータ型の宣言例と、それをprintf文中で用いる例を示します。

データ宣言例

```
char a;
int b;
```

```
long c;
float d;
double e;
```

データ代入例

```
a=120;
b=12345;
c=1234567890;
d=123.456;
e=1234567.890;
```

データ出力例

```
printf("%d¥n",a);      ……char型の数値出力(intと同じ%dを使う)
printf("%d¥n",b);      ……int型の数値出力
printf("%ld¥n",c);     ……long型の数値出力
printf("%f¥n",d);      ……float型の数値出力
printf("%f¥n",e);      ……double型の数値出力(floatと同じ%fを使う)
```

もし読者のCコンパイラが16ビットサイズintであるときは、表現範囲が小さいため、

```
int a;
a=70000;
```

といった大きな数値の代入はおかしな結果になります(オーバーフローという)。このときは表現範囲の大きいlong型を使うことになります。次にlong型変数の使用例を示します。

```
long a;
a=70000;
a=a*2;
printf("a=%ld¥n",a); /* 出力は a=140000 */
```

13.5 演算子の種類

Cには演算子が数多く揃っています。以下にそれを整理します。これを見て、使いたい演算子があれば、その使用例のように記述してみてください。なお、ここではあまり

にも特殊な演算子は除いてあります。

算術演算子

算術演算子は数値を計算するものです。

演算子	説明	使用例
+	正符号(単項プラス演算子)	+a
-	負符号(単項マイナス演算子)	-a
*	乗算	a=b*c;
/	除算	a=b/c;
%	余り	a=b%c;
+	加算	a=b+c;
-	減算	a=b-c;

関係演算子

関係演算子は2つの要素の大小関係や等値関係を判定するものです。

演算子	説明	使用例
<	小さい	if(a<b)
<=	小さいか等しい	if(a<=b)
>	大きい	if(a>b)
>=	大きいか等しい	if(a>=b)
==	等しい	if(a==b)
!=	等しくない	if(a!=b)

論理演算子(! && ||)

論理演算子は真偽値を否定したり、複数の関係演算条件を組み合わせたりするものです。

演算子	説明	使用例	コメント
!	否定	if(!a)	aの値が0なら
&&	論理積(かつ)	if(a==b && c==d)	a==bでかつc==dなら
||	論理和(または)	if(a==b || c==d)	a==bまたはc==dなら

インクリメント、デクリメント演算子

インクリメント、デクリメント演算子は1加算、1減算を行なうものです。

演算子	説明	使用例	普通の書き方
++	1加算	++a; または a++;	a=a+1;
--	1減算	--a; または a--;	a=a-1;

ビット演算子

ビット演算子はビット単位でデータ操作をするものです。

演算子	説明	使用例
&	ビットごとのAND	a=b & 0x7FFF;
|	ビットごとのOR	a=b | 0x8000;
^	ビットごとのXOR	a=b ^ 0x000F;
~	ビットの反転（1の補数）	a= ~a;
<<	左シフト	a=a << 2;
>>	右シフト	a=a >> 2;

代入演算子

通常の代入演算子は"="であり、

```
a=100;
```

のように使われます。

複合代入演算子

通常の代入演算子とは別に、Cには複合代入演算子といわれるものがあります。これを使うと、たとえば、

```
fileposition = fileposition + 20;
```

という記述を、

```
fileposition += 20;
```

と短く書くことができます。

演算子	使用例	一般記法
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;
&=	a &= b;	a = a & b;
^=	a ^= b;	a = a ^ b;
|=	a |= b;	a = a | b;
<<=	a <<= b;	a = a << b;
>>=	a >>= b;	a = a >> b;

多重代入式

代入演算子は多重代入処理することができます。たとえば、

```
a=b=c=10;
```

と書くと、3つの変数すべてに10が代入されます。

カンマ演算子（,）

カンマ演算子は、本来1つしか式を書けないところに複数の式を書くことを可能にします。たとえば、

```
sum=0;
mul=1;
for(i=1; i<=5; i++){
    sum=sum+i;
    mul=mul*i;
}
```

という記述は、カンマ演算子を用いることにより、

```
for(sum=0,mul=1,i=1; i<=5; i++){
    sum=sum+i;
    mul=mul*i;
}
```

とすることができます。

13.6 制御文の種類

Cに備わっているすべての制御文をその利用例で示します。

if

```
if(a==1){        ……aが1なら
    b=2;         }これを実行
    c=3;
}
else{            ……さもなくば
    b=4;         }こちらを実行
    c=5;
}
```

for

```
for(i=1; i<=10; i++){    ……iの値が1から10になるまでループする
    a=a+i;               }その間これを実行
    b=b*i;
}
```

while

```
while(a[i]!=0){          ……a[i]の値が0でない間ループする
    sum=sum+a[i];        }その間これを実行
    i++;
}
```

do-while

do-while文はwhile文と違って、ループの終端部分でループ継続条件を判断します。

```
do{                     ……ループ開始
    c=getchar();        ……1文字読み込み
    putchar(c);         ……それを出力
}while(c!='.');         ……その文字がピリオドでなければループ続行
```

break

break文はforループやwhileループを途中で強制的に打ち切るときに使用します。またswitch文の中でも使用されます。

```
for(i=1; i<=10; i++){      ……iの値が1から10になるまでループする。ただし
    if(dt[i]==0)           ……もしdt[i]の値が0ならこのforループを打ち切る
        break;
    printf("%d¥n",dt[i]);
}
```

continue

continueはforループやwhileループの中で、ある回の処理をスキップさせるのに用いられます。次の記述の実行結果は、

```
1
2
4
5
```

となります(3の出力がない)。

```
for(i=1; i<=5; i++){      ……iの値が1から5になるまでループする。ただし
    if(i==3)              ……iの値が3ならその回の処理はスキップする
        continue;
    printf("%d¥n",i);
}
```

switch

switch文は変数（この例ではa）の値によって多方向分岐するときに用いられます。

```
switch(a){          ……aの値により分岐
case 1:             ……aが1なら
    b=10;           }これを実行
    c=20;
    break;
case 3:             ……aが3なら
    b=30;           }これを実行
    c=40;
    break;
case 8:             ……aが8なら
    b=50;           }これを実行
    c=60;
    break;
default:            ……それ以外なら
    b=70;           }これを実行
    c=80;
}
```

return

returnは関数から値を戻すときに使われます。また関数の任意の位置から戻るときにも用いられます。

```
int max(int a, int b)
{
    if(a>b)              ……a>bなら
        return a;        ……aの値を戻す
    else                 ……そうでないなら
        return b ;       ……bの値を戻す
}
```

goto

goto文は同じ関数の中の任意の場所に直接制御を移します。Cでは他の制御文の機能が充実していますから、通常はgoto文を使うことはありません。

```
void smpf(void)
{
    ...
    ...
    if(a==5) goto abc;          ……もしaが5なら
    ...
    ...
abc:                            ……ここに直接ジャンプする
    ...
}
```

13.7 ポインタ

Cにはポインタと呼ばれる機能があります。変数はデータを記憶しますが、その変数自体はどこかのアドレスにあります。ポインタはこの変数のアドレスを記憶するものです。次にポインタ処理の例を示します。

```
int a,b;                ……int型変数a、bを宣言
int *p;                 ……int型変数の存在するアドレスを記憶できるポインタpを宣言

a=1234;                 ……変数aに1234を設定
b=5678;                 ……変数bに5678を設定

p=&a;                   ……ポインタpに変数aのアドレスを代入
printf("%d¥n",a);       ……出力は1234になる（aの値）
printf("%d¥n",*p);      ……出力は1234になる（pの指すアドレスにある値）

p=&b;                   ……ポインタpに変数bのアドレスを代入
printf("%d¥n",b);       ……出力は5678になる（bの値）
printf("%d¥n",*p);      ……出力は5678になる（pの指すアドレスにある値）
*p=9999;                ……pの指すアドレスにある値を9999にする（結果としてbが9999
                          になる）
```

ここで'&'は変数のアドレスを取りだす演算子、'*'はポインタの指すアドレスにある値を取りだす演算子です。この記述で最初の"*p"は、

――pの指すアドレス（＝変数aのアドレス）にある値を取りだす

ということで、それは結局あらかじめアドレスを設定していた変数aの値を取りだすことになります。このようにポインタを使うと、アセンブラに似たきめ細かい処理が可能になります。変数aが1000番地というアドレスにあると仮定したときの関係を次に示します。

ポインタと変数の関係

13.8 構造体

　Cでは標準のデータ型を組み合わせて、独自のデータグループを作ることができます。たとえば100人分の生徒の国語、英語、数学、理科、社会の点数を管理したいと思ったら、普通の方法では、

```
int kokugo[100],eigo[100],suugaku[100],rika[100],syakai[100];
```

というデータを必要とします。そして出席番号15番の生徒の国語の成績は、

```
kokugo[15]=90;
```

のようにしてデータ設定します。ここで出席番号15番の人の全成績を、計算のために出席番号0の変数領域に代入するとしたら、

```
kokugo[0]=kokugo[15];
eigo[0]=eigo[15];
suugaku[0]=suugaku[15];
rika[0]=rika[15];
syakai[0]=syakai[15];
```

という処理をすることになります。ところがここで、

```
struct{
    int kokugo,eigo,suugaku,rika,syakai;
}tensuu[100];
```

という変数宣言をしたら、データ設定は、

```
tensuu[15].kokugo=90;
```

となり、データの代入処理は、

```
tensuu[0]=tensuu[15];
```

で済んでしまいます。このようにキーワードstructを使った変数構造を「構造体」と呼んでいます。構造体は、関連のある複数のデータを1つの名前で管理できるというメリットがあります。

13.9 プリプロセッサ命令

Cプログラムの先頭部分に書かれる、

```
#include <stdio.h>
```

はプリプロセッサ命令（pre：前もって、process：処理する）と呼ばれるものです。これは内部的に、一般のコンパイル処理に先だって行なわれる処理なので、この名前があります。#includeはコンパイルに先だって、この位置にファイルを読み込むという働きをする、代表的なプリプロセッサ命令です。

もう1つ有用なプリプロセッサ命令に#defineがあります。これは記号定数の設定によく使われます。たとえば、

```
#define PAI 3.14159
```

とすると、以降のプログラム中で3.14159という変な数字の代わりにPAIというわかりやすい名前を使うことができるようになります。たとえば、

```
s=r*r*PAI;
```

とすることができます。また論理処理をするときには、

```
#define TRUE   1
#define FALSE  0
```

という記号定数がよく使われます。このように記述すると以降、TRUEとFALSEという識別子を1と0の代わりに使うことができます。またおなじみのEOFは、stdio.hファイルの中で次のように定義されています。

```
#define EOF (-1)
```

13.10 printfの使い方

printfの基本的な使い方を用例で示します。

変数	出力対象	書き方	説明
int a;	整数	printf("%d¥n",a);	aの値を10進数で出力
int b;	整数	printf("%X¥n",b);	bの値を16進数で出力
int c;	整数	printf("%4d¥n",c);	4桁幅10進数で表示
int d;	文字	printf("%c¥n",d);	dの値を文字コードとみて該当する文字を出力
long e;	整数	printf("%ld¥n",e);	eの値を倍長10進数で表示
float f;	実数	printf("%f¥n",f);	fの値を実数で表示
double g;	実数	printf("%f¥n",g);	gの値を実数で表示
char str[80];	文字列	printf("%s¥n",str);	文字列strを表示

13.11 scanfの使い方

scanfの基本的な使い方を用例で示します。

scanfの使い方

変数	入力対象	書き方	説明
int a;	整数	scanf("%d",&a);	int値をaに入力
long b;	整数	scanf("%ld",&b);	long値をbに入力
float c;	実数	scanf("%f",&c);	float値をcに入力
double d;	実数	scanf("%lf",&d);	double値をdに入力
char str[80];	文字列	scanf("%s",str);	文字列をstrに入力

注：文字配列**str**への入力のときは'**&**'は不要。

Chapter 14

さらなる向上のために

ここでは本書を読んでCに興味をもった方が、さらに次のステップに進むためにはどうすればよいのかを示すことにします。初歩の段階から、C言語をマスターするまでには次のようなステップが考えられます。

1. C言語の入門書を読む

Cコンパイラを購入するとマニュアルがついてきますが、これをいきなり読んで理解できる人は限られています。初心者には、まず市販の入門書を読んでC言語の概略を知ることが理解の近道です。

2. Cコンパイラの使い方を知る

Cコンパイラを購入したら、インストールして動作する状態にしなければなりません。そのうえでエディタの使い方やコンパイラコマンドの使い方をマスターします。これは入門書を読むことと並行して行なえばいいでしょう。

3. 簡単なサンプルプログラムを実際に動作させる

入門書に載っているような短いサンプルプログラムを、実際にエディタで入力して動作させてみます。正しく動作したら、そのプログラムを少し変更して出力の変化を確認したり、わざと間違った記述をして、どのようなエラーメッセージがでるか確認します。

4. 基本的なプログラムをマスターする

データの入出力、演算、条件処理、繰り返し処理などを使った、基本的なプログラムを作成できるようにします。このときもやさしいものから面倒なものへと段階を追って挑戦していきます。

5. 簡単な標準関数を使ってみる

C処理系には標準関数が豊富に揃っています。まずその一覧表に目を通して、どのような便利な関数があるかを調べてみます。最初は簡単に使えそうな関数だけを選んで、実際に使用してみます。見るからに面倒くさそうな関数は、当分使わないようにします。

6. 基本的なファイル処理をマスターする

ファイルから単純にデータを読み込み、ファイルに単純にデータを書き込むような、シンプルな処理手順を勉強します。ファイル処理は面倒なので一度に難しいプログラムに挑戦しないで、はじめは教科書的な例題を使います。そしてその動作を確認したら、それをファイル処理のひな型としてバリエーションを加えて、少しずつ進歩したファイル処理を体験します。

7. Cの全部の仕様に目を通す

基本的なCプログラムが作成できるようになったら、次はCの全部の仕様に目を通します。初心者向けのいかにも入門書といったスタンスの本では、説明不十分な場合があります。中級者向けの本とか、入門書でも厚手で濃い内容の本を求めるようにするとよいでしょう。またマニュアルも積極的に活用するようにします。全部は理解できなくて

も、一通り目を通すことが大切です。

8. ポインタや構造体を使ったプログラムを組めるようにする

初級プログラミングではポインタや構造体は使わなくてもかまいませんが、中上級になると、これは必須の機能になります。特にポインタについてはよく理解しておかないと、おかしな現象がでたとき、どこが問題なのかわからないということになります。

9. 標準関数を使いこなせるようにする

Cの標準関数は機能がとても豊富です。そこですべての標準関数について目を通し、自分のプログラミング分野で利用価値があるかどうかチェックしておきます。標準関数についての把握が十分でないと、用意されている関数なのに知らずに自作した、ということになったりします。

難しい標準関数の場合は、それを使ったサンプルプログラムを見ることが理解に役立ちます。中には何のために用意されたのかよくわからない、という奇妙な関数もあります。これももともとは、

——こういう場面で使うと非常に便利だ

という意図があって開発された関数であることを頭に置き、その「場面」を推測するようにすると理解しやすくなります。もっとも、少し考えてもわからないような奇怪な関数は用途が限られますから、忘れてしまっても実害がないことも少なくありません。

10. プログラム開発手法をマスターする

プログラミングとは、単にソースプログラムを書いて、それをコンパイルするだけではありません。効率よくプログラム開発をするためには、プログラムをモジュールという単位に分割して作成する方法をマスターする必要があります。そのモジュールの数が多くなったら、混乱することがありますので、メイクという管理ツールを使いこなすことも必要な場合があります。またよく使うモジュールをライブラリというものに登録して、いつでも再利用できるようにする方法もあります。デバッグも専用のデバッグツールを使えるようにして、効率的なデバッグが行なえるようにします。

11. 応用プログラムを書けるようにする

C言語の入門書にでてくるプログラムは非常に洗練されています。ここで洗練されているとは、いかにもわかりやすくて、いかにも性格のよいデータが使ってあって、教科書的な模範解答になっているということです。

これは初心者に提示するサンプルプログラムとしては重要なことです。説明しようとする事項以外の要素を削ぎ落としたプログラムのほうが、初心者を迷わせないからです。しかし初級を通り越したプログラマの場合はそうではありません。教科書的なプログラムがスタートにすぎないことを理解し、それを卒業し、どのような問題でもプログラミングできるようにならなければなりません。

また応用プログラムを書けるようになるためには、単にC言語の知識だけでなく、プログラミング周りの知識を必要とします。事務処理プログラムを作成するためには簿記の知識が必要になることがあります。予測プログラムを書くには数学の力を必要としま

す。しかしこのあたりになると、もうC文法の壁は乗り越えており、後は経験を積みながら次第に上達していくものです。

12. アルゴリズムを勉強する

アルゴリズムとはコンピュータで問題を解決するときの、最適な計算方法のことをいいます。中には特許をとるような世界的に有名なアルゴリズムというものもありますが、もっと身近な場面でもアルゴリズムは必要です。あるいはアルゴリズム的な発想が必要です。

たとえば長い配列の中の特定のデータを発見するには、どうやってプログラミングしたら一番早いかとか、ごちゃごちゃした多重ループや、何層にも重なった深いif文をすっきり表現する方法はないかとか、処理を簡潔にするためにあらかじめデータに細工をしておいたらどうか、といったことがあります。よいアルゴリズムを採用すれば、一般的にプログラムの処理時間は短くなります。ソート（並べ替え）のアルゴリズムなどはよく知られています。

13. 得意のパターンを作る

このアルゴリズムは書物を読んで勉強するのも方法ですが、自分が求めている問題解決策が都合よく見つかるとは限りません。その場合、自分でいつも問題意識をもって、よりよい方法を考案するようにします。そうして獲得した方法は、自分の得意の処理パターンとして活用すると生産性が上がります。

このように初心者にとっては気の遠くなるような道ですが、自分はCで何をしたいのかを見極めることも必要です。自分はCの知識をどこまで必要としているのかが明確であれば、それ以上の知識は見切ってしまってかまいません。Cを楽しむとか、簡単なツールを作るとかいう目的であれば、Cの勉強はそれほど負担ではないと思います。

本書の役目はこれで終了です。さらに上達を望む読者は、次のレベルの本を楽しみ、マニュアルの解説を楽しみ、ちょっとレベルアップしたサンプルプログラムを楽しみ、新しい技法を覚えるのを楽しみ、自作のプログラムを楽しみながら、ゆっくりとステップアップを図っていかれたらよいと思います。C言語プログラミングを楽しんでください。

Appendix

Visual C++を利用して
Cコンパイラを使う方法

Cコンパイラの動く環境

読者が実際にCコンパイラを使いたいと思ったとき、それが動作する環境を設定しなければなりません。以前のパソコンはMS-DOSで動くのが大部分でしたから、

——MS-DOS上で動くCコンパイラ

を用意すればよく、それは比較的簡単な話でした。しかし現在のパソコンはWindowsの時代になっています。通常、C言語はWindowsの上で直接動作させることはできません。そのためWindowsパソコンで使用するときは、

——Windows上の［MS-DOSﾌﾟﾛﾝﾌﾟﾄ］の上で動作するCコンパイラ

という環境設定をしてやる必要があります。ここで［MS-DOSﾌﾟﾛﾝﾌﾟﾄ］は、Windowsの上で従来のMS-DOS用プログラムを動作可能にするソフトウェアです。

現実的には、Windows上で使用できるCコンパイラの例としてMicrosoft社のVisual C++があります。この製品はWindows用アプリケーションを開発する目的のC++言語ですが、従来のCコンパイラ部分を内蔵しています（Microsoft Cというものがあります）。ですから、

Windowsパソコン上の［MS-DOSﾌﾟﾛﾝﾌﾟﾄ］の上で
Visual C++の中のMicrosoft Cを使用する

という選択ができることになります。これはWindowsやVisual C++という製品の普及度からみると、

——初心者がC言語の勉強をするための現実的な対応策の1つ

といえます。ここでは参考として、「Visual C++の中のMicrosoft C」を使用可能にするための環境の設定方法を示します。なお、以降の説明において、WindowsやMS-DOSに対する一般的な操作は理解しているものとします。またVisual C++自身のインストールは完了しているものとします。これらの操作がわからない場合は、近くのよく知っている人に聞くか、製品マニュアルを読むようにしてください。

環境変数を設定する

MS-DOS上でCコンパイラを使用する場合、使用するCコンパイラやライブラリの格納場所を示すための環境変数を設定する必要があります。この環境変数の設定のためにVisual C++は、

```
VCVARS32.BAT
```

というバッチプログラムを用意しています。これを実行することで、必要な環境変数が簡単に設定されます。このバッチファイルはVisual C++をインストールしたパソコンの、

～¥bin

というディレクトリにあります。ここで"～"はVisual C++がインストールされた、基になるディレクトリ名を意味します。これはユーザーのインストール時の指定によって変更になる場合があります。たとえば著者の場合、

c:¥program files¥devstudio¥vc¥bin

という名前のディレクトリになっています。

環境変数設定用のメモリを確保する

上に示した例のように、Windowsではディレクトリ名が長くなる傾向にあります。そのため、このVCVARS32.BATバッチファイルを実行すると、

環境変数のための領域が足りません.

というエラーが出ることがあります。この現象の防備のためには、[MS-DOSﾌﾟﾛﾝﾌﾟﾄ]のプロパティ画面で、

[環境変数の初期ｻｲｽﾞ(V)]を1024にする

という設定をしてやります。後で設定画面例を示します。

作業用のディレクトリを作成する

Cプログラムをたくさん作るとファイルが混乱しますので、Cプログラム専用のディレクトリを作るときれいです。私はCWORKというディレクトリを作成しています。ここではその例を示します。また先ほど述べたVCVARS32.BATファイルも、あらかじめこのCWORKディレクトリにコピーしておくと便利です。もともとの位置ではパス名が長すぎて、指定するのが面倒になるからです。コマンドの例を次に示します。

```
C:¥>md ¥cwork          ……作業用ディレクトリを作る

C:¥>cd ¥cwork          ……作業用ディレクトリcworkに移る

C:¥CWORK>copy "c:¥program files¥devstudio¥vc¥bin¥vcvars32.bat"
                       ……vcvars32.batを自ディレクトリにコピーする

C:¥CWORK>vcvars32      ……vcvars32.batを実行する
```

Cコンパイラを使う

一連の準備が完了したら、[MS-DOSﾌﾟﾛﾝﾌﾟﾄ]を起動して、MS-DOS画面に入ります。そしてソースプログラムを作成し、コンパイルすると実行ファイルができます。たとえばWindowsに付属のEDITエディタを使うと、次のような操作になります。

```
C:¥CWORK>edit smp1.c      ……ソースファイルを作成

C:¥CWORK>cl smp1.c        ……それをコンパイル

C:¥CWORK>smp1             ……それを実行
```

これまで述べた一連の設定例を画面で示します。

環境変数の初期サイズを設定(この設定は一度だけでよい)

①[ｽﾀｰﾄ]を右クリックし、これを左クリックする

②この画面になるので、この[ﾌﾟﾛｸﾞﾗﾑ]フォルダをダブルクリックして開く

③これを右クリックする

④これを左クリックする

⑤このプロパティ画面になるので、このタブをクリックする

⑥これを1024にする

⑦これをクリックする

[MS-DOSﾌﾟﾛﾝﾌﾟﾄ] を起動する

Office ﾄﾞｷｭﾒﾝﾄの新規作成
Office ﾄﾞｷｭﾒﾝﾄを開く
ﾌﾟﾛｸﾞﾗﾑ(P)
最近使ったﾌｧｲﾙ(D)
設定(S)
検索(F)
ﾍﾙﾌﾟ(H)
ﾌｧｲﾙ名を指定して実行(R)...
Windows の終了(U)...
ｽﾀｰﾄ
Windows95

Microsoft Visual Basic 5.0
Microsoft Visual C++ 5.0
Microsoft ﾘﾌｧﾚﾝｽ
ｱｸｾｻﾘ
ｽﾀｰﾄｱｯﾌﾟ
Internet Mail
Internet News
Microsoft Excel
Microsoft Exchange
Microsoft NetMeeting
Microsoft Outlook
Microsoft Word
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ
The Microsoft Network
ｲﾝﾀｰﾈｯﾄ ｴｸｽﾌﾟﾛｰﾗ
ｴｸｽﾌﾟﾛｰﾗ
ｵﾝﾗｲﾝ活用ｶﾞｲﾄﾞ

⑧[ｽﾀｰﾄ]を左クリックし、この項目を左クリックして起動する

ディレクトリの設定などをする

⑨CWORKディレクトリを作成する（この操作は一度だけでよい）

⑩CWORKディレクトリに移る

⑪VCVARS32.BATファイルをコピーする（この操作は一度だけでよい）

```
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ
8 x 16

Microsoft(R) Windows 95
   (C)Copyright Microsoft Corp 1981-1996.

C:¥Windows>md ¥cwork

C:¥Windows>cd ¥cwork

C:¥CWORK>copy "¥program files¥devstudio¥vc¥bin¥vcvars32.bat"
        1 個のファイルをコピーしました.

C:¥CWORK>vcvars32
Setting environment for using Microsoft Visual C++ tools.
C:¥CWORK>edit smp1.c_
```

⑫VCVARS32.BATファイルを実行して環境変数を設定する

⑬smp1.cソースファイルを作成する（EDITエディタを使う場合）

ソースプログラムを作成

⑭EDITエディタでプログラムを入力中の画面

```
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ - EDIT
ﾌｧｲﾙ(F)  編集(E)  検索(S)  画面(V)  ｵﾌﾟｼｮﾝ(O)  ﾍﾙﾌﾟ(H)
C:¥cwork¥smp1.c
#include <stdio.h>

main()
{
    printf("abcde¥n");
}

F1=ﾍﾙﾌﾟ    行:1    桁:1
```

コンパイルと実行

⑮作成したソースファイルsmp1.cを見る

⑯smp1.cをコンパイルする

```
MS-DOS ﾌﾟﾛﾝﾌﾟﾄ

C:¥CWORK>type smp1.c
#include <stdio.h>

main()
{
    printf("abcde¥n");
}

C:¥CWORK>cl smp1.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright (C) Microsoft Corp 1984-1997. All rights reserved.

smp1.c
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.

/out:smp1.exe
smp1.obj

C:¥CWORK>smp1
abcde

C:¥CWORK>_
```

⑰smp1.exeを実行する

INDEX

記号

数字

A

B

C

INDEX

INDEX

P

Q

R

S

T

U

V

W

X

あ行

INDEX

か行

さ行

INDEX

た行

な行

は行

INDEX

ま行

や行

ら行

わ行

改訂 新C言語入門 スーパービギナー編

1992年3月19日　初版第1刷発行
1998年6月22日　初版第19刷発行
1998年8月31日　改訂版第1刷発行
2000年3月10日　改訂版第8刷発行

著　者……………………林　晴比古（はやし　はるひこ）
発行者……………………岡崎　眞
発行所……………………ソフトバンク パブリッシング株式会社
〒103-8508　東京都中央区日本橋箱崎町24-1
販売　03（5642）8100
編集　03（5642）8140
組　版……………………ホームデータ
印　刷……………………株式会社　厚徳社

本文デザイン…………日野絵美
本文イラスト…………Suguru.T（Vision Works）
カバーデザイン………火取ユーゴ

乱丁本、落丁本は小社販売局にてお取り替えいたします。定価はカバーに記載されております。
本書の内容に関するご質問などは、ご面倒でも小社テクニカル局DD室まで必ず書面にてご連絡くださいますようお願いいたします。
なお、本書の内容に基づく運用結果については責任を負いかねますので、ご了承ください。

Printed in Japan　　ISBN4-7973-0671-8